Efficient sequence comparison via combining alignment and alignment-free techniques
algorithms and bioinformatics research

Ayad, Lorraine Abdelmasih Khalil

Awarding institution:
King's College London

Efficient Sequence Comparison via Combining Alignment and Alignment-Free Techniques

Algorithms and Bioinformatics Research

Lorraine Abdelmasih Khalil Ayad

Department of Informatics

King's College London

UK

June 2019

Abstract

Sequence comparison is the core computation of many applications involving textual representations of data. Edit distance is the most widely used measure to quantify the similarity of two sequences. Edit distance can be defined as the minimal total cost of a sequence of edit operations required to transform one sequence into the other.

The motivation herein lies specifically within the development of algorithms and their implementation for sequence comparison, avoiding computing alignments under the edit distance measure, where possible. One of the benefits of this work is that it is not necessarily limited to computational biology, but also has applications in image recognition.

The following algorithms were designed to solve and optimise previously presented work found within the literature. This thesis provides an in depth analysis of three algorithms that have been designed, implemented and tested.

The first, hCED is a heuristic solution for computing the cyclic edit distance between two strings using a new distance measure, namely the β-blockwise q-gram distance. The second algorithm, MARS builds on this work, by computing accurate rotations for a given set of circular sequences and outputs the rotated sequences, which can then later be used to compute a multiple sequence alignment. The final algorithm, CNEFinder looks into the analysis of conserved non-coding elements, regions of a genetic sequence found to be evolutionarily conserved across multiple organisms, without needing to compute whole genome alignments.

Acknowledgements

First and foremost I would like to thank my supervisor Dr Solon Pissis for all his hard work and support throughout my PhD. Not only has he dedicated a lot of time and effort through my supervision, but his guidance has enabled me to learn an abundance of skills and earn great experience in a short amount of time. He has allowed my PhD to be a challenging, yet engaging journey, proving to me that it was definitely worthwhile.

I would also like to thank Professor Costas Iliopoulos for his support throughout my PhD as well as providing me with great opportunities for collaboration with other institutes worldwide.

The time and dedication I have spent on my PhD over these few years would not have been possible without the constant care and encouragement of my parents; Abdelmasih Ayad and Magda Ayad, as well as my sister Marina Ayad and brother-in-law Petro Lambros. I am indebted to them for their unconditional love throughout my journey and beyond. I would also like to thank my extended family for consistently asking about me and supporting my progress.

This journey towards my PhD would not have been the same without my friends and members of the Algorithms and Bioinformatics group at King's College London. They have helped to make it truly an enjoyable and memorable time of my life.

I would finally like to acknowledge The Engineering and Physical Sciences Research Council (EPSRC) for their financial support throughout my PhD, who I would not have been able to successfully complete my PhD without.

Publications

1. L.A.K. Ayad, S.P. Pissis and A. Retha, "libFLASM: a software library for fixed-length approximate string matching", BMC Bioinformatics, vol. 17, no. 1, 2016, pp. 454.

2. L.A.K. Ayad, S.P. Pissis,"MARS: improving multiple circular sequence alignment using refined sequences", BMC Genomics, vol. 18, no. 1, 2017, pp. 86.

3. L.A.K. Ayad, C. Barton, S.P. Pissis, "A faster and more accurate heuristic for cyclic edit distance computation", Pattern Recognition Letters, 2017.

4. H. Alamro, L.A.K. Ayad, P. Charalampopoulos, C.S. Iliopoulos, S.P. Pissis, "Longest Common Prefixes with k-Mismatches and Applications", in SOFSEM 2018: Theory and Practice of Computer Science: 44th International Conference on Current Trends in Theory and Practice of Computer Science, Krems, Austria, January 29 - February 2, 2018, Proceedings, A. M. Tjoa et al., Eds., Cham: Springer International Publishing, pp. 636-649.

5. M. Alzamel, L.A.K. Ayad, G. Bernardini, R. Grossi, C. S. Iliopoulos, N. Pisanti, S.P. Pissis, G. Rosone, "Degenerate String Comparison and Applications", in 18th International Workshop on Algorithms in Bioinformatics (WABI 2018), L. Parida, E. Ukkonen, Eds., Dagstuhl, Germany: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, pp. 21:1-21:14.

6. L.A.K. Ayad, S.P. Pissis, D. Polychronopoulos; "CNEFinder: finding conserved non-coding elements in genomes", Bioinformatics, Volume 34, Issue 17, 1 Septem-

ber 2018, Pages i743-i747.

7. L.A.K. Ayad, G. Bernardini, R. Grossi, C.S. Iliopoulos, N. Pisanti, S.P. Pissis, G. Rosone, "Longest Property-Preserved Common Factor", SPIRE, Springer, 2018, pp. 42–49.

8. L.A.K. Ayad, C. Barton, P. Charalampopoulos, C.S. Iliopoulos, S.P. Pissis, "Longest Common Prefixes with k-Errors and Applications", SPIRE, Springer, 2018, pp. 27–41.

9. L.A.K. Ayad, M. Chemillier S.P. Pissis (2018) "Creating improvisations on chord progressions using suffix trees", Journal of Mathematics and Music, 12:3, 233-247.

10. L.A.K. Ayad, G. Badkobeh, G. Fici, A. Heliou & S.P. Pissis, "Constructing Antidictionaries in Output-Sensitive Space", 13 May 2019, 2019 Data Compression Conference (DCC). pp. 538-547.

Contents

1 Introduction

Sequence comparison is the core computation of many applications involving textual representations of data. Edit distance is the most widely used measure to quantify the similarity of two sequences. Edit distance can be defined as the minimal total cost of a sequence of edit operations required to transform one sequence into the other.

The motivation herein lies specifically within the development of algorithms and their implementation for sequence comparison, avoiding computing alignments under the edit distance measure, where possible. One of the benefits of this work is that it is not necessarily limited to computational biology, but also has applications in image recognition.

The following algorithms were designed to solve and optimise previously presented work found within the literature. This thesis provides an in depth analysis of three algorithms that have been designed, implemented and tested. The first, hCED is a heuristic solution for computing the cyclic edit distance between two strings. A cyclic or circular string is one which has no definite beginning or end and can be imagined in such a way where the first and last character of the string are positioned next to each other.

Circular sequences are widely discussed in the literature including within areas of image recognition. Freeman proposed a method for encoding geometric curves so that they can be easily manipulated using existing computational tools [41]. The curves making up an image can be represented using a continuous list of angles which can then be simplified by encoding these values into a numerical list. This was further simplified by using a hexagonal array to represent at most 6 angle variations that the outline of an image can take, such that each numerical value represents an angle which varies at most 60 degrees. An 8-direction chain code can also be used which represents the 8 directions a curve can take to represent the outline of an image [41] as seen in Figure 1.

Example The direction diagram presented in Figure 1a can be used to identify an 8-direction chain-code for the diagram in Figure 1b. Starting at the filled circle and working in a clockwise manner around the image would give the chain-code `0766465432121`.

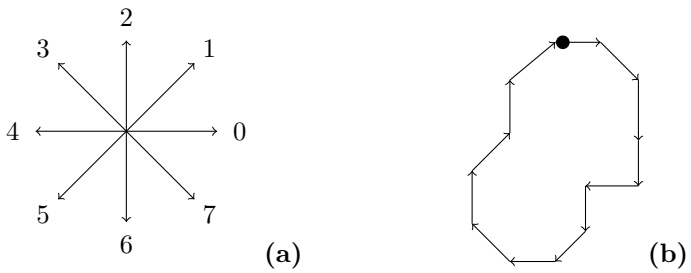


Figure 1: 8 directions a curve can take to represent the outline of an image.

This encoding can be used for pattern recognition where images have a similar closed boundary [90]. Identifying images that have a similar boundary allows them to be classed together which can assist in the process of image restoration [52]. This encoding method has also been used to classify handwritten digits [70] and likewise can be used for the interpretation of chest x-rays and aircraft analysis [90].

Circular structures can also be found in abundance in the biological world. The polyoma virus, a small DNA virus that is widespread in nature [2] obtains this structure [119]. Similarly, bacterial DNA is generally made up of a single circular chromosome [109], as is the genomic structure of archaea [3]. The existence of these structures presents the need for algorithms to be able to analyse these organisms. It is known for example that nucleotide mutations as well as rotations often occur in viruses. Therefore there is a need to determine if a pair of viruses have a mutation due to a circular rotation or from another cause [48]. This can then be linked to a specific disease or gives more understanding about the phylogenetics (evolutionary history) of a group of related organisms [71].

Being able to compute the similarity between circular sequences can aid with the previously defined applications. A measure which is invariant to the starting position of the sequence needs to be used to be able to compute how similar or different a pair of circular sequences are [90]. Several exact and heuristic algorithms exist that can compute this measure, namely the edit distance between a pair of circular sequences. These algorithms are discussed in more detail in Section 3. Due to the great difference in time complexities

when finding exact and heuristic solutions for this problem, it is sometimes beneficial to make use of a heuristic solution which gives near accurate results. The current best performing heuristic in terms of time and accuracy is the weighted Bunke and Buhler algorithm [73]. This algorithm makes use of a dynamic programming matrix called an edit graph made up of a quadratic set of nodes of total size $(|x|+1) \times (2|y|+1)$ where x and y are the input circular sequences. Optimal paths are found on this graph to identify the smallest distance between sequences x and y.

The bottleneck of this algorithm lies within its quadratic time complexity. Our contribution aims to improve on this complexity, specifically for computing solely, an accurate rotation such that the edit distance between two circular sequences is minimal. We present a heuristic algorithm that uses a new distance measure introduced in [47] and is discussed more thoroughly in Section 3.

The second algorithm, MARS builds on this work, by computing accurate rotations for a given set of circular sequences, such as those given in Figure 2, and outputs the rotated sequences, which can then later be used to compute a multiple sequence alignment.

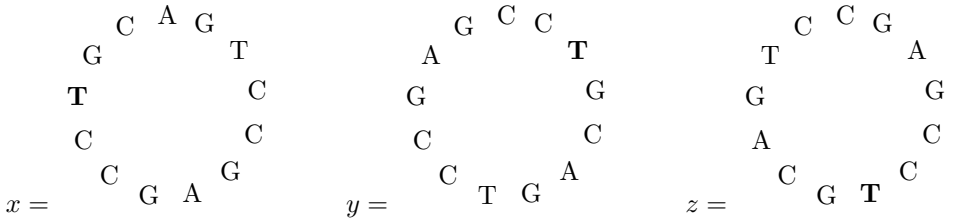


Figure 2: Circular sequences x, y and z.

Example By breaking each of the three circular sequences presented in Figure 2 into linear sequences, by making the top most nucleotide of each x, y and z, the first character of each respective sequence, we obtain the following sequences:

$$x = \texttt{AGTCCGAGCCTGC}$$

$$y = \texttt{CCTGCAGTCCGAG}$$

z = CGAGCCTGCAGTC

However, if broken at certain positions in each sequence, such that those nucleotides presented in bold are the first character of each respective sequence, then it is clear that x, y and z are all identical. $x = y = z =$ TGCAGTCCGAGCC.

The idea of MARS is to accurately compute these rotations, given that the set of input sequences may have been arbitrarily broken at random positions to obtain a set of linear sequences.

As previously described, circular molecular structures are abundant in a range of organisms including bacteria, archaea, eukaryotes and in viruses [25, 50]. DNA replication which leads to cell division is common in eukaryotic cells [55] as well as in bacteria [8]. Mitochondrial DNA, which are commonly found in eukaryotes are circular in structure and are generally conserved across evolution. Replication of mitochondrial DNA occurs frequently [55], which leads to the production of exact copies of the same DNA. Mutations are common during cell replication. This can result in the change of a single base pair, known as a point mutation, or a deletion of a few base pairs which can affect the function of a gene [63].

Multiple sequence alignment, which involves being able to align multiple sequences is key in phylogenetic analysis and the study of evolutionary relationships among species [84]. Molecular sequence alignment allows potential homology among nucleotide or amino acid positions to be discovered, meaning similarity in structure indicates shared evolutionary origin between organisms. This alignment allows for more understanding about the evolutionary divergence between sequences as well as the inference of historical relationships among genes and species [58].

However, to be able to compute a multiple *circular* sequence alignment, an accurate rotation needs to first be identified for each sequence of the input set. To be able to compute this, a measure is required to compute which rotation reduces the overall distance between every sequence pair [17]. The sum-of-pairs score which computes the sum of all

pair-wise induced alignment scores is known to be NP-hard [117]. As a result, several heuristic algorithms have been developed to find an accurate alignment.

Two current state-of-the-art tools that assist in the computation of multiple circular sequence alignments are Cyclope [74] and BEAR [11]. Cyclope computes pairwise alignments using an exact cubic algorithm. The quadratic heuristic algorithm of Cyclope only performs well when all sequences are very similar. On the other hand, BEAR presents two algorithms discussed in more detail in Section 4 where one heuristic is designed for less divergent sequences and the other for more divergent sequences. Experimental results show that BEAR is over 20 times faster than Cyclope [11]. However, the more divergent algorithm makes use of input parameters than can give rise to errors if set to small values. To overcome the drawbacks found in these tools, we present MARS, a heuristic tool based on the pairwise circular sequence comparison algorithm hCED, discussed in Section 3. More details about MARS including experimental analysis can be found in Section 4.

The final algorithm, CNEFinder looks into the analysis of conserved non-coding elements (CNEs), regions of a genetic sequence, sometimes greater than 500 base pairs in length, found to be evolutionarily conserved across multiple organisms. There is very little presented work in the literature that analyses CNEs which are known to be present in all metazoa (multicellcular) genomes; Aparicio et al [5] analysed these elements and have identified their roles in the development of multicellular organisms by acting as enhancers, elements with which proteins interact to transmit signals to genes [15]. It is however known that they do not encode for proteins and their complete functionality is still unknown [88].

As a result, there is a need to be able to further analyse these elements to identify more about their functionality. Alignment based techniques exist which require the pair of input sequences to first be aligned before they are analysed [14]. There also exist several databases that store CNE elements that have been identified from previous research, where solutions have been tailored to the organisms being analysed. These include, but are not limited to those found in the UCNEbase [30]. An example of its interface can be

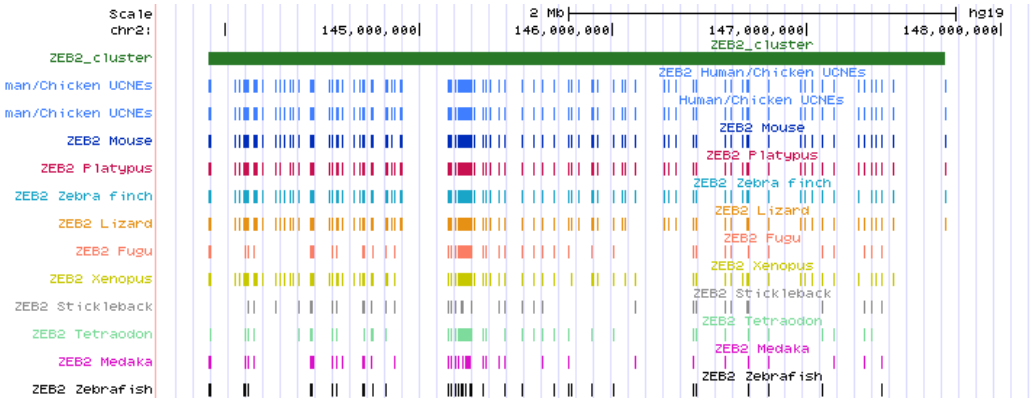


Figure 3: CNEs conserved across a range of organisms, including human, chicken and mouse, retrieved from the UCNEBase.

seen in Figure 3, where it is clear that previously identified CNEs are conserved across a range of organisms.

We present CNEFinder, a tool tailored for CNE identification, given a pair of genetic sequences. It is an alignment-free based tool and also does not require an index of the input sequences to be computed. More details are presented in Section 5 where experimental results show the efficiency and accuracy of CNEFinder.

All three tools were implemented in the C++ programming language and have been thoroughly tested, with results presented in the corresponding sections of this thesis. The implementation of all three tools is distributed under the GNU General Public License and they are freely available at the following locations:

hCED: `https://github.com/lorrainea/hCED`

MARS: `https://github.com/lorrainea/MARS`

CNEFinder: `https://github.com/lorrainea/CNEFinder`

2 Definitions and Notations

We begin with a few definitions to help the reader understand more clearly the algorithms to be discussed.

2.1 Strings

We think of a *string* (or sequence) x of *length* m as an array $x[0 \ldots m-1]$, where every $x[i]$, $0 \leq i < m$, is a *letter* drawn from some fixed *alphabet* Σ of size $|\Sigma| = \mathcal{O}(1)$. The *empty string* of length 0 is denoted by ε. Given string y, a string x is considered a *factor* of y if there exist two strings u and v, such that $y = uxv$. Consider the strings x, y, u, and v, such that $y = uxv$. We call x a *prefix* of y if $u = \varepsilon$; we call x a *suffix* of y if $v = \varepsilon$. When x is a factor of y, we say that x *occurs* in y. Each occurrence of x can be denoted by a position in y. We say that x occurs at the *starting position* i in y when $y[i \,..\, i+m-1] = x$; alternatively we may refer to the *ending position* $i+m-1$ of x in y [28].

2.2 Circular strings

A circular string of length m can be viewed as a traditional linear string which has the left- and right-most letters wrapped around and positioned next to each other. Under this notion, the same circular string can be seen as m different linear strings, which would all be considered equivalent. Given a string x of length m, we denote by $x^i = x[i \ldots m-1]x[0 \ldots i-1]$, $0 < i < m$, the ith *rotation* of x and $x^0 = x$ [28]. Consider, for instance, the string $x = x^0 = \texttt{abababbc}$; which has the following rotations: $x^1 = \texttt{bababbca}$, $x^2 = \texttt{ababbcab}$, and so on. We say that two strings x and y are *conjugate* if there exist two strings u and v such that $x = uv$ and $y = vu$.

2.3 Edit distance

Given a string x of length m and a string y of length $n \geq m$, the *edit distance*, denoted by $\delta_E(x, y)$, is defined as the minimal total cost of edit operations required to transform one string into the other [29]. In general, the allowed operations are as follows:

- *Insertion*: insert a letter in y, not present in x; $(\varepsilon, b),\ b \neq \varepsilon$
- *Deletion*: delete a letter in y, present in x; $(a, \varepsilon),\ a \neq \varepsilon$
- *Substitution*: replace a letter in y with a letter in x; $(a, b),\ a \neq b, \texttt{and}\ a, b \neq \varepsilon$.

By $\mathsf{ins}(b)$, $\mathsf{del}(a)$, and $\mathsf{sub}(a, b)$, $a \neq b$, and $a, b \in \Sigma$, we denote the cost of insertion, deletion, and substitution operations, respectively. In many applications, we only want to count the number of edit operations, considering the cost of each to be 1 [61]. This distance is known as *Levenshtein distance*, a special case of edit distance where unit costs apply.

Example Let $x =$ `AGTCGTACTAGATG`, $y =$ `AGTTCGTACTGCTG` and $\mathsf{ins}(b) = \mathsf{del}(a) = 3$ and $\mathsf{sub}(a, b) = 1$. It is clear from the alignment below that $\delta_E(x, y) = 7$:

```
AG-TCGTACTAGATG
AGTTCGTACT-GCTG
```

2.4 Cyclic edit distance

The *cyclic edit distance* between a circular string x and a circular string y is the minimum edit distance between x and every cyclic shift j of y. It is denoted by $\delta_{\mathrm{CE}}(x, y)$ and is more formally defined as $\delta_{\mathrm{CE}}(x, y) = \min_i(\min_j, \delta_E(x^i, y^j)) = \min_j \delta_E(x, y^j)$ [66].

2.5 q-gram distance

We give some further definitions following [113]. A *q-gram* is defined as any string of length q over alphabet Σ. The set of all q-grams is denoted by Σ^q. For example, the

string AGTACT has the following q-grams of length 3: AGT, GTA, TAC, ACT. The *q-gram profile* of a string x is the vector $G_q(x)$, where $q > 0$ and $G_q(x)[v]$ denotes the total number of occurrences of q-gram $v \in \Sigma^q$ in x.

Definition 1. *Given strings x of length m and y of length $n \geq m$ and an integer $q > 0$, the* q-gram distance $D_q(x, y)$ *is defined as:*

$$\sum_{v \in \Sigma^q} |G_q(x)[v] - G_q(y)[v]| \,. \tag{1}$$

Example Let $x =$ CTCTGAGC, $y =$ TCTCGCGC and $q = 3$. $D_q(x, y) = 8$.

v	CTC	TCT	CTG	TGA	GAG	AGC	TCG	GCG	CGC
$G_q(x)[v]$	1	1	1	1	1	1	0	0	0
$G_q(y)[v]$	1	1	0	0	0	0	1	1	2

2.6 β-blockwise q-gram distance

For a given integer parameter $\beta \geq 1$, [47] defined a generalisation of the q-gram distance by partitioning x and y in β *blocks* as evenly as possible, and computing the q-gram distance between each pair of blocks, one from x and one from y. The rationale is to enforce *locality* in the resulting overall distance. For the sake of presentation in the rest of this thesis, we assume in Section 3 and Section 4 that the lengths $|x| = m$ and $|y| = n$ are both multiples of β, so that x and y are conceptually partitioned into β blocks, each of size m/β for x and n/β for y.

Definition 2. *Given strings x of length m and y of length $n \geq m$ and integers $\beta \geq 1$ and $q > 0$, the* β-blockwise q-gram distance $D_{\beta,q}(x, y)$ *is defined as*

$$\sum_{j=0}^{\beta-1} D_q\left(x\left[\frac{jm}{\beta} \,..\, \frac{(j+1)m}{\beta} - 1\right], y\left[\frac{jn}{\beta} \,..\, \frac{(j+1)n}{\beta} - 1\right]\right). \tag{2}$$

Example Let $x =$ CTCTGAGC, $y =$ TCTCGCGC, $q = 3$ and $\beta = 2$. $D_{\beta,q}(x, y) = 4$.

$j = 0$

v	CTC	TCT
$G_q(x)[v]$	1	1
$G_q(y)[v]$	1	1

$j = 1$

v	GAG	AGC	GCG	CGC
$G_q(x)[v]$	1	1	0	0
$G_q(y)[v]$	0	0	1	1

The example shows strings x and y split into two β-blocks. The first $j = 0$ has a q-gram distance of 0 as all q-grams of length 3 found in $x[0 \dots 3]$ can be found within $y[0 \dots 3]$. The second block, $j = 1$ has a q-gram distance of 4 as all q-grams of length 3 found in $x[4 \dots 7]$ can not be found within $y[4 \dots 7]$ and vice versa. Therefore overall x and y have a β-blockwise q-gram distance of 4.

2.7 Suffix array

Given a string x of length m, a suffix array is an integer array that stores the index position of all lexicographically sorted suffixes of x, which can be constructed using $\mathcal{O}(m)$ time and space [68]. The longest common prefix array stores the length of the longest common prefix between two adjacent suffixes of x as they are stored in the suffix array.

Example Take string $x =$ `ACGTAC$` of length n where the terminating character \$ which is lexicograhically smaller than every other character has been added to the end of x. The suffix array, SA and longest common prefix array, LCP of x are the following:

i	0	1	2	3	4	5	6
$x[i]$	A	C	G	T	A	C	$
SA[i]	6	4	0	5	1	2	3
LCP[i]	0	0	2	0	1	0	0

Given the computed SA and LCP array, we can easily compute which suffixes of a string share the same q-grams. From the given example above, looking at LCP[2] = 2, this tells us that the suffixes starting at positions 0 and 4 share the same q-gram of length 2. By looking at SA[2] = 0 and the suffix which occurs lexicographically before it,

$\mathrm{SA}[1] = 4$, we can identify that the LCP between suffixes $x[0\,..\,n-1]$ and $x[4\,..\,n-1]$ is 2 (`AC`) and this is why they share a q-gram of size 2.

3 hCED - a heuristic for Cyclic Edit Distance computation

The work presented in this section was published by Pattern Recognition Letters: L.A.K. Ayad, C. Barton, S.P. Pissis, "A faster and more accurate heuristic for cyclic edit distance computation", Pattern Recognition Letters, 2017.

3.1 Background

Sequence comparison is a fundamental step in many applications involving textual representations of data. Alignments constitute one of the processes commonly used to compare sequences; they are based on notions of distance or of similarity between strings. Edit distance is the most widely used measure to quantify the similarity (or dissimilarity) of two given sequences. It can be defined as the minimal total cost of a sequence of elementary edit operations required to transform one sequence into the other; for a sequence x of length m and a sequence y of length n, it can be computed in time $\mathcal{O}(mn)$ [28].

In many applications it is common to consider sequences with circular structure: for instance, the orientation of two images or the leftmost position of two linearised circular DNA sequences may be *irrelevant*.

In [70], the authors show that computing the edit distance can be used to classify handwritten digits, where the contours of the digits are represented with an 8-direction chain-code [41]; a sequence over an eight-letter alphabet, representing the eight cardinal directions that the contour faces when following the outline of an image in a clockwise motion.

Example applications where image retrieval is required include digital libraries and multimedia editing [97]. Computing the cyclic edit distance is a key requirement for image processing and shape matching. The contours of a shape may be represented through a cyclic sequence which can be used in the computation of the cyclic edit distance. This can identify similarities in shapes which appear to be distinct from one another [69, 79].

Circular molecular structures are abundant in all domains of life: bacteria, archaea, and eukaryotes, and in viruses. Exhaustive reviews of circular molecular structures can be found in [25] and [50].

Using standard techniques to align circular sequences could incorrectly yield a high genetic distance between closely-related sequences. Indeed, when sequencing molecules, the position where a circular sequence starts can be totally arbitrary. For instance, the linearised human (NC_001807) and chimpanzee (NC_001643) mitochondrial DNA (mtDNA) sequences do not start in the same region [47]. Due to this arbitrariness, a suitable rotation of one sequence would give much better results for a pairwise alignment. This motivates the design of efficient algorithms that are specifically devoted to the comparison of circular sequences [12, 13, 6, 47].

The *cyclic edit distance* problem can be defined as follows. Given a sequence x of length m and a sequence y of length n, find the minimal edit distance between any conjugate (cyclic rotation) of x and any conjugate of y.

Few exact algorithms exist which are able to compute the cyclic edit distance between x and y. Maes designed an elegant divide-and-conquer algorithm which runs in time $\mathcal{O}(mn \log m)$ [66]. The idea of this algorithm is to identify optimal edit paths which do not cross each other on the edit graph of xx and y. An exact branch and bound algorithm based on Maes' algorithm, which runs in time $\mathcal{O}(mn \log m)$, was proposed by Barrachina and Marzal [10]. This method explores only the nodes on the edit graph that could lead to an optimal path, resulting in a much faster algorithm *on average*.

Several heuristic approaches exist for approximating the cyclic edit distance. One of the first ones is the Bunke and Buhler (BBA) algorithm [19]. It estimates a lower bound for the cyclic edit distance by searching for an optimal path in time $\mathcal{O}(mn)$. The extended Bunke and Buhler method (EBBA) computes an estimation of the upper bound for the exact cyclic edit distance, also in time $\mathcal{O}(mn)$ [72]. The weighted Bunke and Buhler algorithm (WeBBA) combines the lower and upper bound estimations, computed by the BBA and EBBA algorithms, to produce an approximation of the cyclic edit distance in

time $\mathcal{O}(mn)$ [73]. It is perhaps the best performing heuristic currently.

Palazon-Gonzalez and Marzal [80] studied the same problem but from the *indexing* point of view for classification and retrieval. Their methods eliminate searching for a distance when it is known that it will be greater than the distance (external bound) to the nearest neighbour. They propose two algorithms. The first one modifies the branch and bound algorithm of [10] by avoiding exploring ranges known to be lower than the lower bound in the branch and bound computation. The second one modifies the BBA algorithm by preventing searching for distances when it is known that the final result will not improve the current external bound.

Our contribution In this paper, we propose hCED, a new heuristic algorithm for cyclic edit distance computation. The first important step of this computation is based on an idea that has not been explored by the previous heuristics; that is, considering q-grams, factors of length q. Informally, the q-gram similarity, defined as a distance in [113], is the number of q-grams shared by the two sequences. Theoretical insight to support the suitability of the algorithm is provided. hCED can be split into the following three main stages:

1. The rotation of x that minimises a generalisation of the q-gram distance between x and y is computed using the algorithm in [47];

2. A refinement of this rotation of x is carried out by examining only some short prefixes and suffixes of the rotation and sequence y;

3. Finally, the edit distance between the refined rotation of x and sequence y is computed.

Our main contribution is an extensive experimental study using both DNA and 8-direction chain-code datasets. These results show that hCED is generally faster, up to one order of magnitude, and more accurate than existing state-of-the-art heuristics. A free open-source implementation of hCED is also made available as opposed to current methods.

3.2 Algorithm hCED

Jokinen and Ukkonen [53] showed the following bound which is directly applicable to the Levenshtein distance.

Lemma 1 ([53]). *Let x and y be strings with Levenshtein distance k. Then at least $|x| + 1 - (k+1)q$ of the $|x| - q + 1$ q-grams of x occur in y.*

We first begin by extending Lemma 1 to non-unit costs.

Lemma 2. *Let x and y be strings with $\delta_E(x, y) = k$, such that $C = \min\{\mathit{ins}(b), \mathit{del}(a), \mathit{sub}(a, b)\}$, for some $C > 1$, $a \neq b$, and $a, b \in \Sigma$. Then at least $|x| + 1 - (\lfloor k/C \rfloor + 1)q$ of the $|x| - q + 1$ q-grams of x occur in y.*

Proof. By assumption we have that $\delta_E(x, y) = k$, and if the edit operations do not have uniform cost, we have that the number of edit operations is less than or equal to $\lfloor k/C \rfloor$. Each edit operation could alter at most q different q-grams and hence the lemma follows. □

Consider the case when $C = \min\{\mathsf{ins}(b), \mathsf{del}(a), \mathsf{sub}(a, b)\}$, $D = \max\{\mathsf{ins}(b), \mathsf{del}(a), \mathsf{sub}(a, b)\}$, and $D - C = \mathcal{O}(1)$. This assumption captures most, if not all, real-world edit-distance-based applications. We claim that the lower bound on the number of q-grams is *good* in the following sense. The number e of edit operations must be $\lfloor k/D \rfloor \leq e \leq \lfloor k/C \rfloor$. For $|x| = |y|$ and $e = (|x| - q + 1)/q$, it is easy to design a string such that each operation alters exactly q q-grams. We can then see that the best bound we can achieve in the above lemma, without some stronger assumptions, is $|x| + 1 - (\lfloor k/D \rfloor + 1)q$ shared q-grams and therefore in such cases, the bound in Lemma 2 is within a constant factor. Note that the choice of $e = (n - q + 1)/q$ is not arbitrary; should e be more than this, the pigeon-hole principle shows that it is not possible to distribute e operations in such a way that each occurs at least q positions apart. This means that each operation can no longer alter exactly q q-grams.

Lemma 3. *Let x and y be two conjugate strings. For a given q, x and y share at most $|x| - q + 1$ q-grams and at least $|x| - 2q + 2$.*

Proof. The first part is trivial. Consider the case when x is a string with a distinct letter per position and $q = 1$. Then x and y share exactly $|x| - q + 1$ distinct q-grams.

For the second part, and by definition, notice that x and y can always be decomposed to x_1x_2 and y_1y_2, respectively, where x_1, x_2, y_1, y_2 are strings, such that $x_1 = y_2$ and $x_2 = y_1$. Then it is not difficult to see that by choosing an appropriate decomposition, each pair, (x_1, y_2) and (x_2, y_1), shares $|x_1| - q + 1$ and $|x_2| - q + 1$ q-grams, respectively. The sum $|x_1| - q + 1 + |x_2| - q + 1$ can be re-written as $|x_1| - q + 1 + (|x| - |x_1|) - q + 1$ which gives $|x| - 2q + 2$. This concludes the proof. □

The small difference of the two bounds shown in Lemma 3 tells us that the q-gram distance is not a *good* distance by itself to recover the rotation of x that minimises the edit distance to y.

Based on the aforementioned remarks, we proceed with designing hCED, a three-stage heuristic algorithm for cyclic edit distance computation. In the first stage, an initial rotation of x is computed using the β-blockwise q-gram distance. In the second stage, a refinement of this rotation is performed; and finally, the edit distance between this refined rotation of x and string y is computed.

3.2.1 Stage 1: Circular sequence comparison with q-grams

Grossi et al [47] presented an exact algorithm, namely saCSC, to compute the β-blockwise q-gram distance between x and y. They use this measure to identify an accurate rotation r of x, such that the edit distance between x^r and y is minimal.

redTheir algorithm is as follows. For a string x of length m and string y of length $n \geq m$, initially, the suffix array [68] and longest common prefix array of string xxy is constructed. A rank is assigned to each suffix which has prefix of at least length q, based on the ordering of the suffix array. This means the first i_0 suffixes which have the same

prefix of length at least q, will all get a rank of 0; the next i_1 suffixes which have the same prefix of length at least q, will all get a rank of 1 and this continues until all suffixes have been given a rank. For all suffixes with a unique prefix of length q, these are assigned a unique value, either a_x or a_y, depending if the suffix exists in string x or y, respectively. This computation takes $\mathcal{O}(m+n)$ time.

New arrays x' and y' are constructed which contain the ranks of each of the $q-$grams found in strings x and y, respectively. Vector $P(y')$ stores the number of occurrences of each rank found in y'. Another vector $diff$ stores the differences between $P(y')$ and the corresponding vector which covers the ranks in a window of length $m-q+1$ letters over string x'. These vectors can be updated in constant time for each window. As the window is shifted to the right, the only two ranks which change the values in $diff$ are the newly visited rank, rightmost in the current window and the leftmost rank of the previous window. This is then computed for every window over the string xx and $diff$ is updated accordingly.

When making use of β blocks, we need to update the information for every $diff_j$, where $0 \leq j < \beta$, for every window of length $m-q+1$ letters. The index position of the start of the window that gives a minimum total of the sum of occurrences found in $diff_j$ is chosen as the rotation of x. This gives an overall time complexity of $\mathcal{O}(\beta m+n)$.

The first stage of hCED is essentially the aforementioned algorithm that exploits q-grams (see Lemma 2). For $\beta = 1$ this corresponds to minimising the standard q-gram distance which is not satisfactory (see Lemma 3); however, the generalisation to β blocks enforces the property of locality.

3.2.2 Stage 2: Refinement

In the second stage, hCED refines rotation x^r and produces a refined rotation, denoted by $x^{r'}$. When in the first stage, the algorithm splits strings x and y into β blocks, it naturally disregards locality within each block. Thus when the initial rotation is produced, it may need to be shifted again slightly to the left or to the right. To this end, we introduce a

new input parameter $0 < P \leq \beta/3$ which defines the length $L = \lfloor P \times \frac{m}{\beta} \rfloor$ of the prefixes and suffixes of x^r and y to be considered by the refinement.

The algorithm proceeds as follows. It creates two new strings x' and y' both of length $3L$. In particular, x' is of the form $x_0^r x_1^r x_2^r$, where x_0^r is the prefix of length L of string x^r; x_1^r is a string of length L consisting only of a letter $\$ \notin \Sigma$; and x_2^r is the suffix of length L of string x^r. The same is done for y using the prefix and suffix of y, resulting in the new string y'.

Each rotation of x' is then compared to y' excluding when a letter of x_1^r (letter $\$$) is found at index 0 of the rotation of x'. Notice that the notion of edit distance is not appropriate here due to the existence of letter $\$$ which denotes a *don't care* letter. We thus rather utilise a notion of similarity between strings, for which equalities between letters are positively valued; inequalities, insertions, and deletions are negatively valued; and comparisons involving letter $\$$ are neither positively nor negatively valued. The search consists then in *maximising* a quantity representative of the similarity between the strings.

To this end we make use of the Needleman-Wunsch algorithm [76] to compute a similarity score for each rotation of string x' and string y'. The rotation of x' which results in the maximum score is chosen as the best rotation, and hence, the final rotation r' of x is computed based on this rotation of x'. Ties are broken arbitrarily. Both x' and y' have length $3L$ resulting in a single Needleman-Wunsch call to have a running time of $\mathcal{O}(L^2)$. As this computation is done exactly $2L$ times, once for each rotation, the overall computation of the refinement takes time $\mathcal{O}(L^3)$.

3.2.3 Stage 3: Edit distance computation

In the third stage, hCED computes the edit distance between strings $x^{r'}$ and y. Myers' bit-vector algorithm is used to compute the edit distance when using unit costs for insertion, deletion, and substitution [75]. Myers' algorithm runs in time $\mathcal{O}(\lceil \frac{m}{w} \rceil\ n)$, where w is the word size of the machine. The standard edit distance algorithm is used when computing the edit distance with non-unit costs. It runs in time $\mathcal{O}(mn)$ [28]. Hence, notice that,

compared to the other heuristics, hCED offers an additional advantage. If one is only interested in the rotation minimising the cyclic edit distance, but not the actual value of the distance, they can use algorithm hCED and skip the third stage, allowing for a much faster computation.

3.3 Analysis

The first two stages of algorithm hCED run in total time $\mathcal{O}(\beta m+n+L^3)$. The parameters β and P can be tailored depending on the application; however our experiments show that setting $\beta = \mathcal{O}(m^{1/3})$ and $P = \mathcal{O}(1)$ performs very well in applications with circular strings. This can be theoretically explained as follows. Notice that β should not be too large to allow some flexibility corresponding to insertions and deletions in the alignment. For P, this is not surprising either as the rationale of the second stage is to refine via examining *only* the leftmost and rightmost blocks of each sequence. These parameter choices imply that the first two stages run in time $\mathcal{O}(m^2 + n)$. The third stage runs in time $\mathcal{O}(\lceil \frac{m}{w} \rceil n)$ with unit costs and in time $\mathcal{O}(mn)$ with non-unit costs. The space complexity is $\mathcal{O}(\beta m + n)$; the edit distance and Needleman-Wunsch algorithms can both be implemented in $\mathcal{O}(m + n)$ space [28].

3.4 Experimental Results

Algorithm hCED was implemented in C++ as a program to compute an approximation of the cyclic edit distance. Given two sequences x and y in multiFASTA format, the number of β blocks, and the length q of the q-grams, hCED finds an approximation of the rotation of x that minimises its edit distance from y. It can also output the corresponding rotation of x. The implementation is distributed under the GNU General Public License, and it is available freely at `http://github.com/lorrainea/hCED`.

Another program[1] was used to produce experimental results for the Maes, Branch and Bound, BBA, EBBA, and WeBBA algorithms and compare their performance against that

[1] Obtained through personal communication with author – Guillermo Peris.

of algorithm hCED. The experiments were conducted on a computer using an Intel Core i3-5005U CPU at 2.00GHz under GNU/Linux. Both programs were compiled with g++ version 4.8.5 at optimisation level 3 (O3). All input datasets referred to in this section are publicly maintained at the same website.

Myers' bit-vector algorithm was implemented using the Edlib library [125]. The standard edit distance algorithm was also implemented to show how the hCED algorithm compares to the other heuristics when both unit and non-unit costs are used for the edit distance operations.

3.4.1 Synthetic Data

DNA datasets were simulated using INDELible [39], which produces sequences in a multiFASTA file. A rate for insertions, deletions, and substitutions are defined by the user to vary the similarity of the sequences. 8-direction chain-code datasets were also generated using a simple script that generates random (uniform distribution) sequences over $\Sigma = \{0, 1, \ldots, 7\}$. All datasets used in the experiments are denoted in the form $A.B.C$, where A represents the number of sequences in the dataset; B the average length of the sequences; and C the percentage of dissimilarity between the sequences. The dissimilarity values of 5, 20, and 35 were used for both the DNA data and chain-codes.

Nine datasets were simulated to measure the accuracy for DNA sequences. Each dataset had a varying number of sequences, all with an average length of $2,500$. For each dataset, the algorithms were run for every possible pair of sequences in the set. Three 8-direction chain-code datasets were also produced. These datasets consisted of twelve sequences in each set with an average length of 500. Similarly, for each dataset, the algorithms were run for every possible pair of sequences in the set. For all datasets, we made use of the following parameter values for algorithm hCED: $q = 5$, $\beta = m/50$, and $P = 1.0$.

The values for the cyclic edit distance for each pairwise comparison were computed using the heuristic algorithms. These values were then compared with those output from

the exact cyclic edit distance algorithm by Maes. The number of accurate values output by the heuristic algorithms in comparison to the exact values for the cyclic edit distance were computed as an average percentage, which we define as accuracy.

DNA				
Accuracy (%)				
Dataset	hCED	BBA	WeBBA	EBBA
12.2500.5	**100.000**	83.302	**100.000**	**100.000**
12.2500.20	**100.000**	76.043	99.939	99.905
12.2500.35	**100.000**	77.673	99.933	99.889
25.2500.5	**100.000**	84.798	99.997	99.968
25.2500.20	**99.975**	74.606	99.903	99.868
25.2500.35	**99.961**	73.478	99.882	99.849
50.2500.5	**100.000**	85.303	99.999	99.960
50.2500.20	**99.999**	79.903	99.977	99.940
50.2500.35	**99.981**	74.043	99.910	99.867
8-direction chain-code				
Accuracy (%)				
Dataset	hCED	BBA	WeBBA	EBBA
12.500.5	**100.000**	82.511	99.895	99.401
12.500.20	**100.000**	81.344	99.718	99.481
12.500.35	**100.000**	87.364	99.783	99.586

Table 1: Accuracy of heuristic algorithms in comparison to exact results produced by Maes' algorithm for datasets using unit costs. The highest accuracy for each dataset is shown in bold.

The results in Table 1 show the accuracy of the algorithms for both data types when unit costs were used for insertions, deletions and substitutions. In some applications, in particular in bioinformatics, the cost for insertions and deletions is set higher than the cost for substitutions. Table 2 shows the accuracy for each of the data types when non-unit costs of 3, 3, and, 1 were used for insertion, deletion, and substitution, respectively. It becomes evident from these results that algorithm hCED is the most accurate

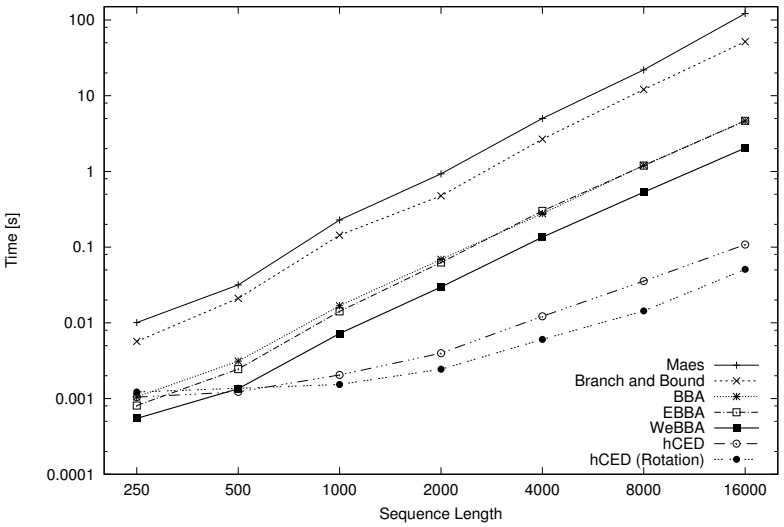
DNA				
Accuracy (%)				
Dataset	hCED	BBA	WeBBA	EBBA
12.2500.5	**100.000**	79.476	**100.000**	**100.000**
12.2500.20	**99.958**	61.197	99.793	99.763
12.2500.35	**99.997**	73.360	99.953	99.909
25.2500.5	**99.986**	80.950	99.981	99.956
25.2500.20	**99.970**	70.523	99.903	99.864
25.2500.35	**99.942**	65.091	99.865	99.831
50.2500.5	**99.996**	81.131	99.992	99.955
50.2500.20	**99.987**	75.358	99.972	99.937
50.2500.35	**99.969**	69.339	99.932	99.888
8-direction chain-code				
Accuracy (%)				
Dataset	hCED	BBA	WeBBA	EBBA
12.500.5	**99.967**	38.044	99.677	99.282
12.500.20	**99.175**	44.569	98.859	98.592
12.500.35	**99.771**	57.554	99.082	98.854

Table 2: Accuracy of heuristic algorithms in comparison to exact results produced by Maes' algorithm for datasets using non-unit costs. The highest accuracy for each dataset is shown in bold.

in comparison to the other heuristic algorithms.

To measure the time performance for both data types, seven pairs of sequences of varying length were simulated. The running time for all sequence pairs were computed ten times and the average was taken. For these experiments, the following parameter values for algorithm hCED were used: $q = 5$, $\beta = \min(50, \sqrt{m})$, and $1.0 \leq P \leq 2.0$. Figure 4 shows the time performance of hCED when using unit costs compared to the other heuristic and exact algorithms. It is clear that as the sequence length grows, hCED is an order of magnitude faster than WeBBA, the current fastest performing algorithm. Notice that hCED is three orders of magnitude faster than Maes' algorithm.

DNA



8-direction chain-code

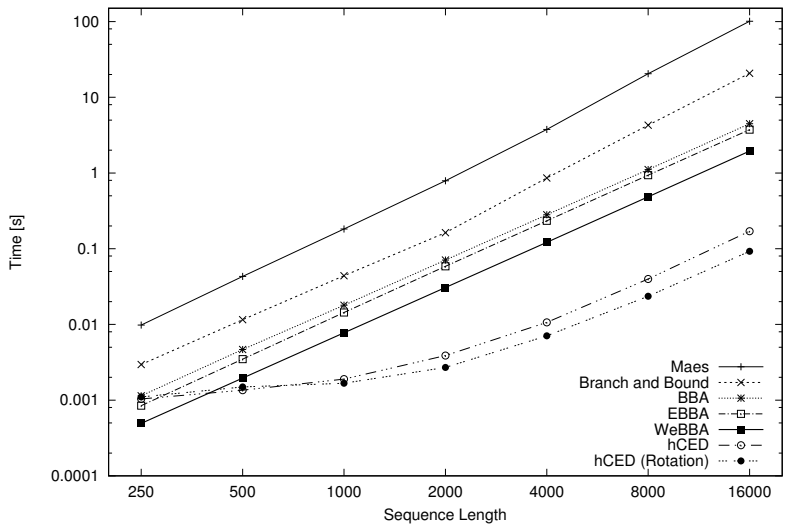
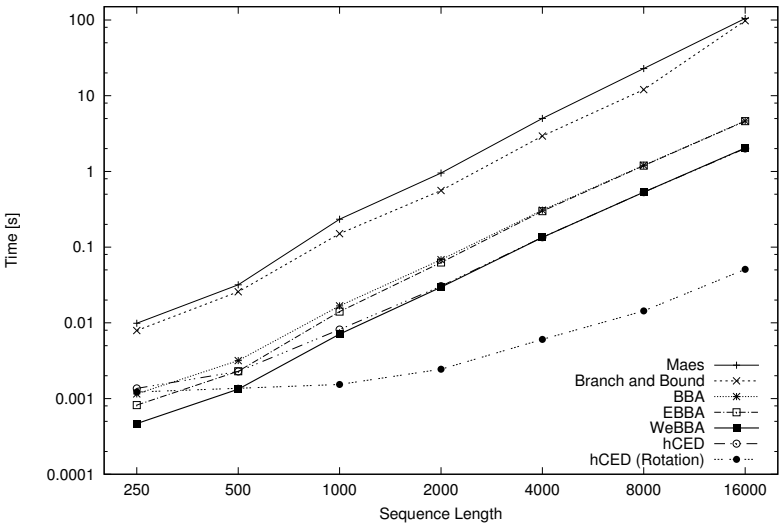


Figure 4: Elapsed time to execute datasets using unit costs

DNA



8-direction chain-code

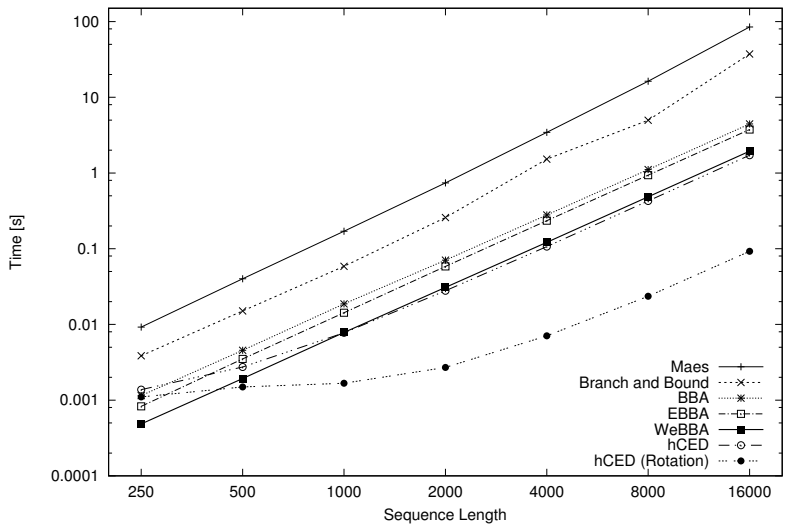


Figure 5: Elapsed time to execute datasets using non-unit costs

Figure 5 shows the time performance of algorithm hCED for both data types when using non-unit costs. A comparison between hCED and the other cyclic edit distance algorithms can be seen in the figure. As the sequence length grows, hCED and WeBBA become the fastest performing algorithms. Both figures also show the time performance of algorithm hCED when *only* the rotation is computed; the cyclic edit distance value is *not* computed. It is evident that dismissing the computation of the cyclic edit distance greatly improves the time performance of hCED. The results of hCED confirm our theoretical analysis.

3.4.2 Real Data

Three datasets made up of nucleotide sequences were used to test the hCED algorithm's ability to identify accurate rotations. The first dataset (Mammals) includes 12 mtDNA sequences of mammals, the second dataset (Primates) includes 16 mtDNA sequences of primates, and the last one (Viroids) includes 18 viroid RNA sequences. The average sequence length for Mammals is $16,777$ base pairs (bp), for Primates is $16,581$ bp, and for Viroids is 363 bp.

Table 3 shows the accuracy of hCED's computation of the cyclic edit distance for each pair, which we denote by AP, in comparison to the other heuristics, as well as the average time taken to do so. The experiment was carried out when using unit costs for insertions, deletions, and substitutions, as well as when using the same non-unit costs previously presented. For the Mammals and Primates datasets, we made use of the following parameter values for algorithm hCED: $q = 5$, $\beta = m/100$, and $P = 1.0$. For the Viroids dataset, in which the sequences are much shorter, the following parameters were used instead: $q = 5$, $\beta = m/25$, and $P = 1.0$. It is evident from Table 3, that not only does hCED give the most accurate results, but it is also faster for sequences of long length when using both unit and non-unit costs.

Handwritten digits from the MNIST database [60] were also used and sorted into ten sets. Each image was placed in one of ten datasets, depending on the value of the drawn

Unit Costs								
Program	hCED		BBA		WeBBA		EBBA	
Dataset	AP (%)	Time (s)	AP (%)	Time (s)	AP (%)	Time (s)	AP (%)	Time (s)
Mammals	**99.618**	**0.479**	69.477	4.870	96.482	2.162	96.469	5.015
Primates	**99.743**	**0.202**	74.256	4.766	98.749	2.115	98.742	4.904
Viroids	**98.363**	0.003	61.057	0.002	97.874	**0.001**	97.614	0.002
Non-unit Costs								
Program	hCED		BBA		WeBBA		EBBA	
Dataset	AP (%)	Time (s)	AP (%)	Time (s)	AP (%)	Time (s)	AP (%)	Time (s)
Mammals	**98.221**	**2.092**	57.092	4.870	86.728	2.202	86.716	5.012
Primates	**99.672**	**1.964**	65.859	4.748	94.443	2.175	94.431	4.898
Viroids	**98.155**	0.003	49.746	0.002	97.623	**0.001**	97.288	0.002

Table 3: Accuracy of heuristic algorithms in comparison to exact results produced by Maes' algorithm and elapsed-time comparison for real nucleotide data. The highest accuracy and fastest time for each dataset are shown in bold.

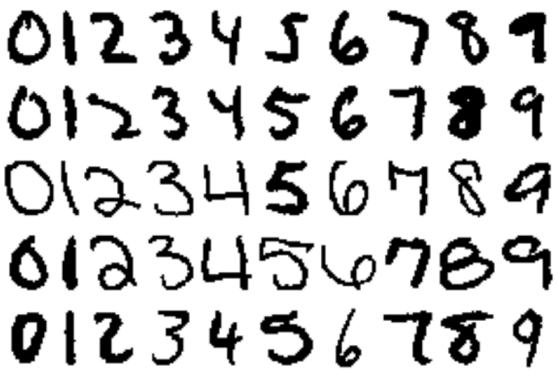


Figure 6: Handwritten digits

digit. Each handwritten digit was in the form of a 28×28 matrix consisting of pixel values. $5,000$ of the $60,000$ images were extracted and converted into binary matrices. A normalised 8-direction chain-code was produced for the handwritten digits, where a subset can be found in Figure 6. Normalising the chain-code allows the image to be treated as a circular sequence of minimum magnitude. This produces a sequence independent of the rotation of the image. This was calculated by identifying the number of direction changes

between two adjacent elements of the chain-code in an anticlockwise direction (see [45], for details).

	Accuracy (%) - Unit Costs				Accuracy (%) - Non-unit Costs			
Dataset	hCED	BBA	WeBBA	EBBA	hCED	BBA	WeBBA	EBBA
0.479.55	**91.781**	52.908	90.126	88.271	**87.699**	41.323	80.256	78.535
1.563.43	**93.159**	58.589	92.037	89.629	**88.213**	49.656	85.895	83.727
2.488.73	**94.504**	54.442	90.265	88.860	**89.395**	41.645	77.193	76.145
3.493.80	**95.371**	54.043	89.441	88.277	**87.396**	38.844	72.854	71.849
4.535.69	**93.855**	59.600	90.351	89.022	**89.163**	45.332	77.621	76.350
5.434.78	**95.287**	53.118	87.869	86.640	**85.338**	37.622	69.817	68.870
6.501.60	**94.139**	54.796	88.954	87.328	**86.279**	38.630	72.112	70.759
7.550.65	**92.436**	55.092	88.984	88.485	**88.276**	41.018	75.323	74.108
8.462.56	**94.006**	55.841	90.720	89.023	**87.597**	41.948	75.948	74.398
9.495.54	**94.211**	55.946	89.814	88.029	**86.950**	41.765	76.159	74.715

Table 4: Accuracy of heuristic algorithms in comparison to exact results produced by Maes' algorithm for handwritten digits. The highest accuracy for each dataset is shown in bold.

Table 4 shows the results of using algorithm hCED to compute the cyclic edit distance for successive pairs in each dataset. Each dataset is in the form $D.E.F$, where D represents the drawn digit; E the number of sequences in the set; and F the average length of the sequences in the set. For these datasets, we made use of the following parameter values for algorithm hCED: $q = 5$, $7 \leq m/\beta \leq 15$, depending on the average length of the sequence, and $P = 1.0$. It is evident from Table 4, that for all sets, hCED is the most accurate when using both unit and non-unit costs. Running times are not presented for the handwritten digits datasets as the sequence lengths are very small.

3.5 Conclusion

In this section, algorithm hCED, a new heuristic approach to approximate the cyclic edit distance, was presented. It is an extension of the q-gram based algorithm presented

in [47] adapted for cyclic edit distance computation. Our main contribution is an extensive experimental study to compare hCED against existing state-of-the-art heuristics for the same problem. In particular, we showed that the performance of hCED, in terms of accuracy and speed, outperforms existing heuristics using both DNA and 8-direction chain-code data.

The inherent structure of hCED allows for two important properties: (i) hCED enables the user to compute only the rotation of x (or an approximation of it) that minimises the cyclic edit distance from y, performing even faster if the actual value for the cyclic edit distance is not required; and (ii) this also enables the usage of the fastest known algorithm for the edit distance computation with unit costs if the actual value for the cyclic edit distance is required. Figure 4 greatly reflects these advantages in practical terms.

Our improvements are particularly important for applications in image retrieval and molecular biology. For instance, algorithm hCED can now be directly used for computing the cyclic edit distance between all pairs of sequences for progressive multiple circular sequence alignment, as seen in the next section.

4 MARS - computing Multiple circular sequence Alignments using Refined Sequences

The work presented in this section was published by BMC Genomics: L.A.K. Ayad, S.P. Pissis,"MARS: improving multiple circular sequence alignment using refined sequences", BMC Genomics, vol. 18, no. 1, 2017, pp. 86.

4.1 Background

The one-to-one mapping of a DNA molecule to a sequence of letters suggests that sequence comparison is a prerequisite to virtually all comparative genomic analyses. Due to this, sequence comparison has been used to identify regions of similarity which may be a byproduct of evolutionary, structural, or functional relationships between the sequences under study [38]. Sequence comparison is also useful in fields outside of biology, for example, in music analysis [21] or pattern recognition [67] as seen in Section 3. Several techniques exist for sequence comparison; alignment techniques consist of either *global* alignment [76, 46] or *local* alignment [99] techniques. Alignment-free techniques also exist; they are based on measures referring to the composition of sequences in terms of their constituent patterns [114]. Pairwise sequence alignment algorithms analyse a pair of sequences, commonly carried out using dynamic-programming techniques [46]; whereas multiple sequence alignment (MSA) involves the simultaneous comparison of three or more sequences.

The purpose of an MSA is to obtain an alignment of the sequences which will represent and give more information about their evolutionary, functional or structural relationship. The alignment, which can be represented as a 2D matrix is obtained by inserting gaps within the sequences to allow identical or similar nucleotides to be aligned in each column of the matrix. In terms of evolution, these gaps represent insertions and deletions within the genome which are known to have occurred during the evolution from a common ancestor [24].

Analysing multiple sequences simultaneously is fundamental in biological research and MSA has been found to be a popular method for this task. One main application of MSA is to find conserved patterns within protein sequences [123] and also to infer homology between specific groups of sequences [58]. MSA may also be used in phylogenetic tree reconstruction [84] as well as in protein structure prediction [98].

Example Consider the following sequences:

Before alignment:	After alignment:
s_0: TCTACCAGAAA	s_0: -TCTACCAGAAA
s_1: TTTAGAGACA	s_1: -TTTA-GAGACA
s_2: TCTTAGGAGGAA	s_2: TCTTAGGAGGAA

The sequences on the right show an MSA of those on the left. Gaps have been inserted to improve the similarity of each column in the alignment.

Let $\mathsf{cost}(\mathtt{a}, \mathtt{b})$ denote the cost of transforming one character in the MSA into another. Let $\mathsf{cost}(\mathtt{a}, \mathtt{-}) = \mathsf{cost}(\mathtt{-}, \mathtt{a}) = -2$, $\mathsf{cost}(\mathtt{a}, \mathtt{b}) = -1$, $\mathsf{cost}(\mathtt{a}, \mathtt{a}) = 1$ and $\mathsf{cost}(\mathtt{-}, \mathtt{-}) = 0$. The sum-of-pairs score (SP-score) computes a score for an MSA based on the pair-wise sum for each column. The first column in the alignment above has a pair-wise score of $\mathsf{cost}(\mathtt{-}, \mathtt{-})$ + $\mathsf{cost}(\mathtt{T}, \mathtt{-})$ + $\mathsf{cost}(\mathtt{T}, \mathtt{-}) = 0 + -2 + -2 = -4$. The SP-score for this MSA is 1.

Using a generalisation of the dynamic-programming technique for pairwise sequence alignments works efficiently for MSA for only up to a few short sequences. Specifically, MSA with the SP-score criterion is known to be NP-hard [117]; and, therefore, heuristic techniques are commonly used [110, 35, 78], which may not always lead to optimal alignments. As a result, suboptimal alignments may lead to unreliable tree estimation during phylogenetic inference. To this end, several methods aimed to have shown that removing unreliable sites (columns) of an alignment may lead to better results [106].

Several discussions of existing filtering methods provide evidence that the removal of blocks in alignments of sufficient length leads to better phylogenetic trees. These filtering methods take a variety of mathematical and heuristic approaches. Most of the methods

are fully automated and they remove entire columns of the alignment. A few of these programs, found in [104, 22], are based on site-wise summary statistics. Several filtering programs, found in [32, 57, 26, 122, 82], are based on mathematical models. However, experimental results found in [106] oppose these findings, suggesting that generally, not only do the current alignment filtering methods not lead to better trees, but there also exist many cases where filtering worsened the trees significantly.

Circular molecular structures can be composed of both amino and nucleic acids [25]. The most common examples of such structures in eukaryotes are mtDNA. mtDNA is generally conserved from parent to offspring and replication of mtDNA occurs frequently in animal cells [55]. This is key in phylogenetic analysis and the study of evolutionary relationships among species [84]. Several other example applications exist including MSA of viroid or viral genomes [18] and MSA of naturally-occurring circular proteins [120].

A fundamental assumption of all widely-used MSA techniques is that the left- and right-most positions of the input sequences are relevant to the alignment. However, the position where a sequence starts (left-most) or ends (right-most) can be totally arbitrary due to a number of reasons: arbitrariness in the linearisation (sequencing) of a circular molecular structure; or inconsistencies introduced into sequence databases due to different linearisation standards. In these cases, existing MSA programs, such as Clustal Ω [96], MUSCLE [34], or T-Coffee [78], may produce an MSA with a higher average pairwise distance than the expected one for closely-related sequences. As previously mentioned, the published human (NC_001807) and chimpanzee (NC_001643) mtDNA sequences do not start in the same genetic region [37]. It may be more relevant to align mtDNA based on gene order [42], however, the tool we present in this paper may be used to align sequences of a broader type. Hence, for a set of input sequences, a solution for these inconsistencies would be to identify a suitable rotation (cyclic shift) for each sequence; the sequences output would in turn produce an MSA with a lower average pairwise distance.

Due to the abundance of circular molecular structures in nature as well as the potential presence of inconsistencies in sequence databases, it becomes evident that multiple circular

sequence alignment (MCSA) techniques for analysing such sequences are desirable. Since MCSA is a generalisation of MSA it is easily understood that MCSA with the SP-score criterion is also NP-hard. To this end, a few programs exist which aim to improve MCSA for a set of input sequences. These programs can be used to first obtain the best-aligned rotations, and then realign these rotations by using conventional alignment programs, such as Clustal Ω, MUSCLE, or T-Coffee. Note that unlike other filtering programs, these programs do not remove any information from the sequences or from their alignment: they merely refine the sequences by means of rotation.

The problem of finding the optimal (linear) alignment of two circular sequences of length m and $n \geq m$ under the edit distance model can be solved in time $O(nm \log m)$ [66]. The same problem can trivially be solved in time $O(nm^2)$ with substitution matrices and affine gap penalty scores [46]. To this end, alignment-free methods have been considered to speed-up the computation [47, 27]. The more general problem of searching for a circular pattern in a text under the edit distance model has also been studied extensively [12], and an average-case optimal algorithm is known [13].

Progressive multiple sequence alignments can be constructed by generalising the pairwise sequence alignment algorithms to profiles, similar to Clustal Ω [96]. This generalisation is implemented in Cyclope [74], a program for improving multiple circular sequence alignment. The cubic runtime of the pairwise alignment stage becomes a bottleneck in practical terms. Other fast heuristic methods were also implemented in Cyclope, but they are only based on some (e.g. the first two) sequences from the input dataset.

Another approach to improve MCSA was implemented in CSA [37]; a program that is based on the generalised circular suffix tree construction. The authors present a trie structure that stores all the rotations of a set of sequences. The best-aligned rotations are found based on the largest chain of non-repeated blocks that belong to all sequences. Unfortunately, CSA is no longer maintained; it also has the restriction that there can be only up to 32 sequences in the input dataset, and that there must exist a unique block that occurs in every sequence.

BEAR [11] is another program aimed to improve MCSA computation in terms of the inferred maximum-likelihood-based phylogenies. The authors presented two methods; the first extends an approximate circular string matching algorithm for conducting approximate circular dictionary matching. A matrix M is outputted from this computation. For a set of d input sequences $s_0, \ldots, s_{d-1}$, M holds values e and r between circular sequences s_i and s_j, where $M[i,j].e$ holds the edit distance between the two sequences and $M[i,j].r$ holds the rotation of sequence s_i which will result in the best alignment of s_i with s_j. Agglomerative hierarchical clustering is then used on all values $M[i,j].e$, to find sufficiently good rotations for each sequence cluster. The second method presented is suitable for more divergent sequences. An algorithm for fixed-length approximate string matching is applied to every pair of sequences to find most similar factors of fixed length. These factors can then determine suitable rotations for all input sequences via the same method of agglomerative hierarchical clustering.

Our contribution We design and implement MARS, a new heuristic method for improving Multiple circular sequence Alignment using Refined Sequences. MARS is based on a non-trivial coupling of the cyclic edit distance algorithm presented in Section 3 with the classic progressive alignment paradigm [51]. Experimental results presented here, using real and synthetic data, show that MARS improves the alignments and outperforms state-of-the-art methods both in terms of accuracy and efficiency. Specifically, to support our claims, we analyse these results with respect to standard genetic measures as well as with respect to the inferred maximum-likelihood-based phylogenies. For instance, we show here that the average pairwise distance in the MSA of a dataset of widely-studied mtDNA sequences is reduced by around 5% when MARS is applied before the MSA is performed.

4.2 Algorithm MARS

We present MARS, a heuristic algorithm for improving MCSA using refined sequences. For a set of d *input* sequences $s_0, \ldots, s_{d-1}$, the task is to *output* an array R of size d such that $s^{R[i]}$, for all $0 \leq i < d$, denotes the set of rotated sequences, which are then input into the preferred MSA algorithm to obtain an improved alignment. MARS is based on a three-stage heuristic approach:

1. Initially a $d \times d$ matrix M holding two values e and r per cell, is computed; where $M[i,j].e$ holds the edit distance between sequences $s_i^{M[i,j].r}$ and s_j. Intuitively, we try to compute the value r that minimises e, that is, the cyclic edit distance.

2. The neighbour-joining clustering method is carried out on the computed distances to produce a *guide tree*.

3. Finally, progressive sequence alignment using refined sequences is carried out using the sequence ordering in the guide tree.

4.2.1 Stage 1. Pairwise cyclic edit distance

In this stage, we make use of a heuristic method for computing the cyclic edit distance between two strings. This method is described in more detail in Section 3, where the β-blockwise q-gram distance between two circular sequences x and y is computed. Specifically, the algorithm finds the rotation r of x such that the β-blockwise q-gram distance between x^r and y is minimal.

The second step of this stage involves a refinement of the rotation for a pair of sequences, to obtain a more accurate value for r. An input parameter $0 < P \leq \beta/3$ is used to create refined sequences of length $3 \times P \times \frac{m}{\beta}$ using x^r and y, where m is the length of x^r. The first refined sequence is $x_0^r x_1^r x_2^r$: x_0^r is a prefix (of P out of β blocks) of string x^r; x_1^r is a string of the same length as the prefix consisting only of letter $\$ \notin \Sigma$; and

x_2^r is a suffix (of P out of β blocks) of string x^r. The same is done for string y, resulting in a refined sequence of the same form $y_0y_1y_2$. Note that large values for P would result in long sequences, improving the refinement of the rotation, but slowing down the computation. A score is calculated for all rotations of these two smaller sequences using Needleman-Wunsch [76] or Gotoh's algorithm [46], making use of substitution matrices for nucleotides or amino acids accordingly. The rotation with the maximum score is identified as the new best-aligned rotation and r is updated if required.

The final step of this stage involves computing the edit distance between the new pair of refined sequences. For unit costs, this is done using Myers' bit-vector algorithm [75] in time $O(\lceil \frac{m}{w} \rceil n)$, where w is the word size of the machine. For non-unit costs this is computed using the standard dynamic programming solution for edit distance [29] computation in time $O(mn)$. Hence, for a dataset with d sequences, a $d \times d$ matrix M is populated with the edit distance e and rotation r for each pair of sequences.

Remark for Stage 1 The simple cost scheme used in Stage 1 for the pairwise cyclic edit distance is sufficient for computing fast approximate rotations. A more complex (biologically relevant) scoring scheme is used in Stage 3 for refining these initial rotations. A yet more complex scoring scheme, required for the final MSA of the sequences output by MARS, can be carried out later on by using any MSA program, and is therefore beyond the scope of the aim of this work.

4.2.2 Stage 2. Guide tree

The guide tree is constructed using Saitou and Nei's neighbour-joining algorithm [91], where a binary tree is produced using the edit distance data from matrix M. Figure 7 shows an example of such a tree, a phylogenetic tree where the evolvement of organisms from common ancestors can be visually identified through the branching structure of the tree [43]. The nodes at the leaves represent sequences corresponding to the genetic coding of each individual organism.

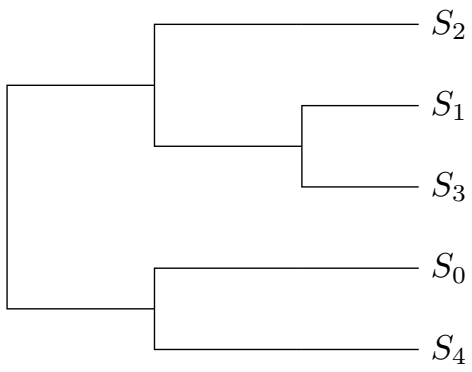


Figure 7: Phylogenetic tree obtainable from neighbour-joining algorithm.

4.2.3 Stage 3. Progressive Alignment

The guide tree is used to determine the ordering of the alignment of the sequences. Three types of alignments may occur:

- Case 1: A sequence with another sequence;
- Case 2: A sequence with a profile;
- Case 3: A profile with another profile;

where a *profile* is an alignment viewed as a sequence by regarding each column as a letter [110]. We also need to extend the alphabet to $\Sigma' = \Sigma \cup \{-\}$ to represent insertions or deletions of letters (gaps). For the rest of this stage, we describe our method using the Needleman-Wunsch algorithm for simplicity although Gotoh's algorithm is also applicable.

For Case 1, where only two sequences are to be aligned, note that rotation r has been previously computed and stored in matrix M during Stage 1 of the algorithm. These two sequences are aligned using Needleman-Wunsch algorithm and stored as a new profile made up of the alignment of two individual sequences which now include gaps. In this case, for two sequences s_i and s_j, we set $R[i] := M[i, j].r$ and $R[j] := 0$, as the second sequence is left unrotated.

The remaining two cases of alignments are a generalisation of the pairwise circular sequence alignment to profiles. In the alignment of a pair of sequences, matrix M provides a reference as to which rotation r is required. In the case of a sequence and a profile (Case 2), this may also indirectly be used as we explain below.

As previously seen, when two sequences s_i and s_j are aligned, one sequence s_j remains unrotated. This pair then becomes a profile which we will call *profile A*. Given the same occurs for another pair of sequences, *profile B* is created also with one unrotated sequence, $s_{j'}$. When *profile A* is aligned with *profile B*, another profile, *profile C* is created. In this case, only the sequences in *profile B* are rotated to be aligned with *profile A*. This results in s_j to be left unrotated in *profile C* where s_j previously occurred in *profile A*. Given a sequence s_k to be aligned with *profile C*, this sequence has a current rotation of 0 as has not yet been aligned with another sequence or a profile. We can identify which rotation is needed to rotate sequence s_k to be aligned with *profile C*, by using the single rotation $M[k, j].r$.

The same condition applies when aligning two profiles (Case 3). All sequences in *profile B* will need to be rotated to be aligned with *profile A*. However, once a single sequence s_j in *profile A* as well as a single sequence $s_{j'}$ in *profile B* with $r = 0$ have been identified, in this case $s_{j'}$ has already been aligned with other sequences. This means gaps may have been inserted and $M[j', j].r$ will no longer be an accurate rotation. By counting the total number g of individual gaps inserted in $s_{j'}$, between positions 0 and the single rotation $M[j', j].r$ of $s_{j'}$, the new suitable rotation for *profile B* would be $M[j', j].r+g$.

Example Consider the following sequences:

s_0: `TAGTAGCT`

s_1: `AAGTAAGCTA`

s_2: `AAGCCTTTAGT`

s_3: `AAGTAAGCT`

s_4: `TTAATATAGCC`

Let *profile A* be:

s_0: A-G-C-TTA-GT

s_1: AAG-C-TAAAGT

s_2: AAGCCTTTA-GT

Let *profile B* be:

s_3: A---AGTAAG-C-T

s_4: A-ATA-TA-GCCTT

Profile C:

s_0: A-G-C-TT-A--GT

s_1: AAG-C-TA-A-AGT

s_2: AAGCCTTT-A--GT

s_3: AAG-C-TA---AGT

s_4: A-GCCTTA-ATA-T

By looking at the original set of sequences, it is clear s_2 in *profile A* and s_3 in *profile B* have a rotation of 0. The other sequences have been rotated and aligned with the remaining sequences in their profile. It is also clear from the original sequences that $M[3,2].r = 4$. When aligning *profile B* with *profile A*, the rotation of $r = 4$ is no longer accurate due to gaps inserted in s_3. As $g = 3$ gaps have been inserted between positions 0 and r of sequence s_3, the final accurate rotation for *profile B* is $M[3,2].r + g = 4 + 3 = 7$ (see *profile C*). □

In the instance when a sequence is to be aligned with a profile or two profiles are to be aligned, a generalisation of the Needleman-Wunsch algorithm is used, similar to that by [116], to compute the alignment score. *Profile A* will always hold the largest number of sequences, allowing *profile B* with fewer sequences to be rotated. We now describe how we can compute a more accurate rotation than that obtained from matrix M.

A frequency matrix F is stored, which holds the frequency of the occurrence of each letter in each column in *profile A*. Equation 3 shows the calculation of frequency matrix F, with all values initialised to 0. $F[c,i]$ holds the frequency of letter c occurring in column i of *profile* $A[0\,..\,|A|-1, 0\,..\,m-1]$ of length m and size $|A|$, where $|A|$ denotes the number of sequences in *profile A*:

$$F[c,i] = F[c,i] + \frac{1}{|A|} \qquad \begin{array}{l} 0 \leq i < m \\ \forall c \text{ in } A[0\,..\,|A|-1,i] \end{array} \tag{3}$$

Equation 4 shows the scoring scheme used for each alignment, where $S[i,j]$ holds the alignment score for column i in *profile A* and column j in *profile B*. gA is the cost of inserting a gap into *profile A* and gB likewise for *profile B*. Matrix S is initialised in the same way as in the Needleman-Wunsch algorithm; and $sim(B[k,j],c)$ denotes the similarity score between letter $c \in \Sigma'$ and the letter at column j of row k (representing sequence s_k) in *profile B*. These similarity scores are computed using the BLOSUM62[2] scoring matrix for protein sequences and the EDNAFULL[3] scoring matrix for DNA sequences.

$$S[i,j] = \max \begin{cases} S[i-1,j-1] + pScore(i,j) \\ S[i-1,j] + gB \\ S[i,j-1] + gA \end{cases} \tag{4}$$

$$pScore(i,j) = \frac{\sum_{c \in \Sigma'} sim(B[k,j],c) \times F[c,i]}{|B|} \quad 0 \leq k < |B|$$

This scoring scheme can be applied naïvely for *profile A* and every rotation of *profile B* to find the maximum score, equating to the best-aligned rotation. However, as information about rotations has already been computed in Stage 1, we may use only some part of *profile B* to further refine these rotations. This refinement is required due to the heuristic computation of all pairwise cyclic edit distances in Stage 1 of the algorithm. To this end, we generalise the second step of Stage 1 to profiles. This step of Stage 1 involves a refinement of the rotation for a pair of sequences via considering only the two ends of each sequence, to create two refined sequences. Similarly here we generalise this idea to refine the rotation for a pair of profiles via considering only the two ends of each profile, to recreate the profiles into profiles with refined sequences. The rotation r with

[2] `ftp://ftp.ncbi.nih.gov/blast/matrices/BLOSUM62`
[3] `ftp://ftp.ncbi.nih.gov/blast/matrices/NUC.4.4`

the maximum score according to the aforementioned scoring scheme is identified as the best-aligned rotation and array R is updated by adding r to the current rotation in $R[i]$, for all s_i in profile B.

4.3 Experimental Results

MARS was implemented in the C++ programming language as a program to compute the rotations (cyclic shifts) required to best align a set of input sequences. Given a set of d sequences in multiFASTA format, the length ℓ of the β blocks to be used, the length q of the q-grams, and a real number P for the refinement, MARS computes an array R according to the algorithm described in Section 4.2. There is also a number of optional input parameters related to Gotoh's algorithm, such as the gap opening and extension penalty scores for pairwise and multiple sequence alignment. A different substitution matrix can be used for scoring nucleotide or amino acid matches and mismatches. The output of MARS is another multiFASTA file consisting of d refined sequences, produced using the rotations computed in R. The output of MARS can then be used as input to the preferred MSA program, such as Clustal Ω, MUSCLE, or T-Coffee.

The implementation is distributed under the GNU General Public License, and it is available freely at `http://github.com/lorrainea/mars`. Experimental results were also produced for Cyclope and BEAR to compare their performance against MARS. The experiments were conducted on a computer using an Intel Core i5-4690 CPU at 3.50GHz under GNU/Linux. All programs were compiled with g++ version 4.8.5 at optimisation level 3 (O3).

4.3.1 Synthetic Data

DNA datasets were simulated using INDELible [39], which produces sequences in a multiFASTA file. A rate for insertions, deletions, and substitutions are defined by the user to vary similarity between the sequences. All datasets used in the experiments are de-

noted in the form $A.B.C$, where A represents the number of sequences in the dataset; B the average length of the sequences; and C the percentage of dissimilarity between the sequences. Substitution rates of 5%, 20%, and 35% were used to produce the datasets under the Jukes and Cantor (JC69) [54] substitution model. The insertion and deletion rates were set to 4% and 6% respectively, relative to a substitution rate of 1.

Nine datasets were simulated to evaluate the accuracy of the proposed method. Each dataset consists of a file with a varying number of sequences, all with an average length of $2,500$ base pairs (bp). The files in Datasets $1-3$ each contain 12 sequences. Those in Datasets $4-6$ each contain 25 sequences; and Datasets $7-9$ contain 50 sequences. All input datasets referred to in this section are publicly maintained at the MARS website.

For all datasets, we made use of the following values for the mandatory parameters of MARS: $q = 5$, $\ell = 50$, and $P = 1.0$. Table 5 shows the results for the synthetic datasets made up of three files which each contained 12 sequences (Datasets $1-3$). The second column shows results for the original datasets aligned using Clustal Ω. All sequences in these datasets were then randomly rotated, denoted in Table 5 by $A.B.C$.rot. The third column shows the results produced when MARS was first used to refine the sequences in the $A.B.C$.rot dataset, to find the best-aligned rotations; and then aligned them again using Clustal Ω. The fifth and sixth columns show likewise using MUSCLE to align the sequences. Tables 6 and 7 show the results produced for Datasets $4-6$ and $7-9$, respectively.

To evaluate the accuracy of MARS seven standard genetic measures were used: the length of the MSA; the number of polymorphic sites (PM sites); the number of transitions and transversions; substitutions, insertions, and deletions were also counted; as well as the average distance between each pair of sequences in the dataset (AVPD). The fourth and last column show the difference between the genetic measures computed for the original datasets and those computed for the datasets rotated using MARS. Note that a decrease in the number of polymorphic sites and insertions and deletions shows an improvement in the alignment.

Program	Clustal Ω	MARS+Clustal Ω		MUSCLE	MARS+MUSCLE	
Dataset 1	12.2500.5	12.2500.5.rot	Difference	12.2500.5	12.2500.5.rot	Difference
Length	2,503	2,503	0	2,503	2,503	0
PM Sites	698	698	0	689	689	0
Transitions	3,845	3,849	+4	3,804	3,804	0
Transversions	4,245	4,251	+6	4,205	4,205	0
Substitutions	12,254	12,264	+10	12,111	12,111	0
Indels	360	360	0	388	388	0
AVPD	191	191	0	189	189	0
Dataset 2	12.2500.20	12.2500.20.rot	Difference	12.2500.20	12.2500.20.rot	Difference
Length	2,662	2,664	+2	2,674	2,674	0
PM Sites	2,228	2,230	+2	2,155	2,155	0
Transitions	16,819	16,502	-317	16,171	16,184	+13
Transversions	15,374	15,719	+345	14,422	14,422	0
Substitutions	47,754	47,799	+45	44,261	44,280	+19
Indels	10,545	10,707	+162	8,817	8,815	-2
AVPD	883	886	+3	804	804	0
Dataset 3	12.2500.35	12.2500.35.rot	Difference	12.2500.35	12.2500.35.rot	Difference
Length	2,526	2,528	+2	2,528	2,528	0
PM Sites	2,062	2,070	+8	2,045	2,045	0
Transitions	18,385	18,167	-218	18,362	18,362	0
Transversions	17,642	17,728	+86	17,316	17,316	0
Substitutions	54,573	54,533	-40	53,807	53,807	0
Indels	2,403	2,575	+172	2,253	2,253	0
AVPD	863	865	+2	849	849	0

Table 5: Standard genetic measures for Datasets $1-3$

Remark for accuracy The use of standard genetic measures to test the accuracy of MARS with synthetic data is sufficient. This is due to the fact that the main purpose of this test is not to check whether we obtain an MSA that is biologically relevant. The ultimate task here was to show that when MARS is applied on the $A.B.C$.rot datasets before MSA is performed we obtain MSAs whose standard genetic measures are similar or even identical to the MSAs of the $A.B.C$ datasets (and this cannot occur by chance) when using the *same* MSA program.

The results show indeed that MARS performs extremely well for all datasets. This

Program	Clustal Ω	MARS+Clustal Ω		MUSCLE	MARS+MUSCLE	
Dataset 4	25.2500.5	25.2500.5.rot	Difference	25.2500.5	25.2500.5.rot	Difference
Length	2,515	2,515	0	2,515	2,515	0
PM Sites	1,243	1,238	−5	1,230	1,230	0
Transitions	20,438	20,422	−16	20,353	20,353	0
Transversions	20,672	20,587	−85	20,498	20,498	0
Substitutions	61,780	61,523	−257	61,289	61,289	0
Indels	2,582	1,932	−650	1,842	1,842	0
AVPD	214	211	−3	210	210	0
Dataset 5	25.2500.20	25.2500.20.rot	Difference	25.2500.20	25.2500.20.rot	Difference
Length	2,600	2,595	−5	2,590	2,591	+1
PM Sites	2,585	2,577	−8	2,572	2,572	0
Transitions	105,738	105,596	−142	106,070	106,256	+186
Transversions	104,778	104,451	−327	103,335	103,238	−97
Substitutions	313,329	312,311	−18	309,953	310,056	+103
Indels	20,524	20,658	+134	13,678	13,784	+106
AVPD	1,112	1,109	−3	1,078	1,079	+1
Dataset 6	25.2500.35	25.2500.35.rot	Difference	25.2500.35	25.2500.35.rot	Difference
Length	2,726	2,751	+25	2,722	2,716	−6
PM Sites	2,700	2,727	+27	2,684	2,679	−5
Transitions	101,801	102,471	+670	104,001	103,796	−205
Transversions	104,993	104,632	−361	100,595	101,078	+483
Substitutions	310,597	311,468	+871	304,100	304,481	+381
Indels	47,080	58,288	+11,2018	35,956	35,110	−846
AVPD	1,192	1,232	+40	1,133	1,131	−2

Table 6: Standard genetic measures for Datasets $4 - 6$

can be seen through the high similarity between the measurements for the original and the refined datasets. Notice that, in particular with MUSCLE, we obtain an *identical or less* average pairwise distance in 8 out of 9 cases between the original and the refined datasets produced by using MARS, despite the fact that we had first randomly rotated all sequences (compare the $A.B.C$ to the $A.B.C$.rot columns).

RAxML [101], a maximum-likelihood-based program for phylogenetic analyses, was used to identify the similarity between the phylogenetic trees inferred using the original and refined datasets. A comparison with respect to the phylogenetic trees obtained using

Program	Clustal Ω	MARS+Clustal Ω		MUSCLE	MARS+MUSCLE	
Dataset 7	50.2500.5	50.2500.5.rot	Difference	50.2500.5	50.2500.5.rot	Difference
Length	2,524	2,524	0	2,524	2,524	0
PM Sites	1,875	1,882	+7	1,861	1,861	0
Transitions	86,781	87,190	+409	86,628	86,628	0
Transversions	91,334	91,584	+250	91,040	91,040	0
Substitutions	262,804	263,687	-117	261,248	261,248	0
Indels	11,531	10,771	−760	8,231	8,231	0
AVPD	223	224	+1	219	219	0
Dataset 8	50.2500.20	50.2500.20.rot	Difference	50.2500.20	50.2500.20.rot	Difference
Length	2,576	2,580	+4	2,582	2,582	0
PM Sites	2,568	2,573	+5	2,575	2,575	0
Transitions	284,302	284,667	+365	282,638	282,670	+32
Transversions	283,651	284,673	+1,022	279,451	279,462	+11
Substitutions	852,738	855,055	+2,317	842,564	842,672	+108
Indels	39,273	45,769	+6,496	33,371	33,369	−2
AVPD	728	735	+7	715	715	0
Dataset 9	50.2500.35	50.2500.35.rot	Difference	50.2500.35	50.2500.35.rot	Difference
Length	2,675	2,697	+22	2,679	2,667	−12
PM Sites	2,675	2,696	+21	2,678	2,666	−12
Transitions	424,910	423,592	−1,318	426,230	426,063	−167
Transversions	431,453	428,874	−2,579	423,113	422,916	−197
Substitutions	1,282,515	1,278,286	−4,229	1,267,683	1,267,699	+16
Indels	92,060	97,398	+5,338	73,890	72,718	−1,172
AVPD	1,122	1,123	+1	1,095	1,094	−1

Table 7: Standard genetic measures for Datasets $7 - 9$

MUSCLE and RAxML was made between the alignment of the original datasets and that of the datasets produced by refining the $A.B.C$.rot datasets using MARS, BEAR, and Cyclope. The Robinson–Foulds (RF) metric was used to measure the distance between each pair of trees. The same parameter values were used for MARS: $q = 5$, $\ell = 50$, and $P = 1.0$. The fixed-length approximate string matching method with parameter values $w = 40$ and $k = 25$ under the edit distance model, were used for BEAR, where w is the factor length used and k is the maximum distance allowed. Parameter v was used for Cyclope to compute, similar to MARS, a tree-guided alignment.

Dataset	BEAR	Cyclope	MARS
12.2500.5	0.000	0.000	0.000
12.2500.20	0.000	0.000	0.000
12.2500.35	0.000	0.000	0.000
25.2500.5	0.000	0.000	0.000
25.2500.20	0.000	0.000	0.000
25.2500.35	0.000	**0.045**	0.000
50.2500.5	**0.021**	0.000	0.000
50.2500.20	0.000	0.000	0.000
50.2500.35	0.000	0.000	0.000

Table 8: Relative RF distance between trees obtained with original and refined datasets. Non-zero values shown in bold.

Table 8 shows that the *relative* RF distance between the original datasets and those refined with MARS is 0 in all cases, showing that MARS has been able to identify the best-aligned rotations, with respect to the inferred trees, for all nine datasets, outperforming BEAR and Cyclope, for which we obtain non-zero values in some cases.

4.3.2 Real Data

In this section we present the results for three datasets used to evaluate the effectiveness of MARS with real data. The first dataset (Mammals) includes 12 mtDNA sequences of mammals, the second dataset (Primates) includes 16 mtDNA sequences of primates, and the last one (Viroids) includes 18 viroid RNA sequences. All input datasets referred to in this section are publicly maintained at the MARS website. The average sequence length for Mammals is $16,777$ bp, for Primates is $16,581$ bp, and for Viroids is 363 bp.

Table 9 shows the results from the original alignments and the alignments produced after refining these datasets with MARS. It is evident that using MARS produces a significantly better alignment for these real datasets, which can specifically be seen through the results produced by aligning with MUSCLE. For instance, the average pairwise distance in the MSA of Primates is reduced by around 5% when MARS is applied before MSA is

performed with MUSCLE.

Program	Clustal Ω	MARS+ Clustal Ω	Difference	MUSCLE	MARS+ MUSCLE	Difference
Mammals						
Length	19,452	18,829	−623	19,784	19,180	−604
PM Sites	12,913	12,265	−648	13,076	12,454	−622
Transitions	135,380	137,589	+2,209	135,794	137,835	+2,041
Transversions	81,945	84,188	+2,243	76,894	78,067	+1,173
Substitutions	295,684	302,331	+6,647	282,608	286,747	+4,139
Indels	82,494	59,348	−23,146	91,164	71,042	−20,122
AVPD	5,729	5,479	−250	5,663	5,421	−242
Primates						
Length	18,176	17,568	−608	18,189	17,669	−520
PM Sites	11,086	10,450	−636	11,023	10,454	−569
Transitions	259,921	261,995	+2,074	262,179	264,245	+2,066
Transversions	100,708	102,336	+1,628	95,403	96,010	+607
Substitutions	439,929	445,252	+5,323	429,532	432,993	+3,461
Indels	80,851	52,727	−28,124	82,117	55,525	−26,592
AVPD	4,339	4,149	−190	4,263	4,070	−193
Viroids						
Length	566	498	−68	486	476	−10
PM Sites	555	484	−71	475	459	−16
Transitions	7,567	7,485	−82	9,338	9,101	−237
Transversions	5,837	5,998	+161	5,491	5,393	−98
Substitutions	19,436	19,291	−145	20,828	20,374	−454
Indels	19,003	18,383	−620	14,323	13,491	−832
AVPD	251	246	−6	229	221	−8

Table 9: Standard genetic measures for real data

Since time-accuracy is a standard trade-off of heuristic methods, in order to evaluate the time performance of the programs, we compared the resulting MSA along with the time taken to produce it using BEAR, Cyclope, and MARS with MUSCLE. Parameter values $w = 100$ and $k = 60$ were used to measure accuracy for the Mammals and Primates datasets for BEAR; $w = 40$ and $k = 25$ were used for the Viroids dataset. Parameter v was used for Cyclope to compute a tree-guided alignment. The following parameter values were used to test the Mammals and Primates datasets for MARS: $q = 5$, $\ell = 100$, and $P = 2.0$; $q = 4$, $\ell = 25$, and $P = 1.0$ were used to test the Viroids dataset.

Table 10 shows the time taken to execute the datasets; for the sake of succinctness,

Table 10 only presents the average pairwise distance measure for the quality of the MSAs. The results show that MARS has the best time-accuracy performance: BEAR is the fastest program for two of the three datasets, but produces very low-quality MSAs; Cyclope is very slow but produces much better MSAs than BEAR; and MARS produces better MSAs than both BEAR and Cyclope and is more than four times faster than Cyclope.

Program	BEAR		Cyclope		MARS	
Dataset	AVPD	Time (s)	AVPD	Time (s)	AVPD	Time (s)
Mammals	5,517	262.96	5,422	1,367.17	5,421	333.50
Primates	4,167	465.17	4,080	2,179.68	4,070	463.25
Viroids	232	0.30	223	1.44	221	0.82

Table 10: Elapsed-time comparison using real data

A common reliability measure of MSAs is the computation of the Transitive Consistency Score (TCS) [23]. The TCS has been shown to outperform existing programs used to identify structurally correct portions of an MSA, as well as programs which aim to improve phylogenetic tree reconstruction [24]. BEAR, Cyclope, and MARS were used to identify the best rotations for the sequences in the Viroids dataset; the output of each, as well as the unrotated dataset was then aligned using MUSCLE. The following TCS was computed for the Viroids dataset when unrotated: 260, as well as when rotated with BEAR, Cyclope, and MARS, respectively: 249, 271, and 293. The same was done using Clustal Ω to align the output sequences, with a TCS of 249 for the unrotated dataset. The following scores were computed for the rotated dataset in the respective order: 233, 244, and 269. These results show that when using two different MSA programs, MARS obtains a higher TCS than the unrotated dataset in both cases, outperforming BEAR and Cyclope, which do not always obtain a higher TCS compared to that of the unrotated dataset.

4.4 Conclusion

A fundamental assumption of all widely-used MSA techniques is that the left- and right-most positions of the input sequences are relevant to the alignment. This is not always the case in the process of MSA of mtDNA, viroid, viral or other genomes, which have a circular molecular structure.

We presented MARS, a new heuristic method for improving Multiple circular sequence Alignment using Refined Sequences. Experimental results, using real and synthetic data, show that MARS improves the alignments, with respect to standard genetic measures and the inferred maximum-likelihood-based phylogenies, and outperforms state-of-the-art methods both in terms of accuracy and efficiency.

5 CNEFinder - Finding conserved non-coding elements in genomes

The work presented in this section was published by Oxford Bioinformatics: L.A.K. Ayad, S.P. Pissis, D. Polychronopoulos; "CNEFinder: finding conserved non-coding elements in genomes", Bioinformatics, Volume 34, Issue 17, 1 September 2018, Pages i743-i747.

5.1 Background

CNEs are a pervasive class of elements that are usually identified by inspecting whole-genome alignments between two or more genomes. CNEs can be extremely conserved across evolution, yet they do not encode for proteins. Some of these elements play roles in the development of multicellular organisms acting as enhancers [5]. Although they can be referred to in the literature with different names (UCEs, UCNEs, CNS, to name a few), the prevailing view is that these sets of elements are largely overlapping, with their genesis, functions and evolutionary dynamics being largely unknown.

The authors in [86] provide an introduction to the characteristics of CNEs and their known functionality so far. CNEs were identified as early as only three decades ago, where studies identified hundreds of thousands of elements that had maintained $> 70\%$ sequence similarity over millions of years of evolution. Since then, several studies of CNEs have been carried out which have been tailored to specific groups of organisms.

CNEs do not have a random distribution across genomes, but are known to form clusters around genes. Analysis of these elements has shown that many of the identified CNEs serve as regulatory elements that are important in the early stages of vertebrate development and brain formation. Also, when compared to surrounding elements of a genome, CNEs have been found to be AT-rich in general [88]. These elements have also been identified in plants [62] which raises more queries about the functionality of CNEs in general.

As very little is known about the role of CNEs within genomes, further analysis is

required to understand more. For the research carried out on these elements so far, it can be seen that CNE identification methods may be classified into two major categories: alignment-based and alignment-free methods.

Alignment-based methods. Alignment-based methods identify CNEs by inspecting pairwise or multiple whole-genome alignments. Several tools exist that generate whole-genome alignments, such as BLASTZ [95], MULTIZ [16], and LASTZ [49]. For a pair of sequences, CNEs are defined as elements which satisfy specific length and sequence identity percentage thresholds [33, 14, 92]. The threshold values depend on the evolutionary distance between species under comparison. Not all CNEs identified by whole-genome comparisons of mammalian genomes appear conserved when the same conservation criteria is used on more distant genome comparisons. Thus, those thresholds are somewhat arbitrary.

Alignment-free methods. Alignment-free methods avoid some of the problems associated with whole-genome alignments, such as computational complexity, highly fragmented assemblies, and inflexibility. Variants of BLAST are usually used in the homology search on repeat- and coding sequence-masked genomes [9]. [118] proposed an alignment-free method based on k-mers. All k-mers occurring a single time in the reference genome are mapped to the species of interest with a short-read aligner and then overlapping hits are merged into longer CNEs. This approach increases the sensitivity of CNE detection by overcoming the ambiguities and errors in the alignment, such as gap insertions and occurrences of a split across alignment blocks. However, this approach incurs a small false positive rate due to mishandling of hits with multiple copies, either from genome duplications or assembly errors. Most importantly, and similar to many other cases, the authors do not make their implementation for identifying CNEs publicly available.

CNE databases. There also exist many CNE databases which contain already precomputed sets of CNEs: Ancora [36], CEGA [31], cneViewer [83], CONDOR [121], UCbase [64],

UCNEbase [30], and VISTA [115]. On the one hand, this highlights the importance of this research topic among the biological community. On the other hand, these databases are static and seldom updated. Furthermore, the sets of CNEs stored in these databases are identified using custom scripts, written in different programming languages, and tailored to the biological needs of each study.

Our contribution In summary, we would like to stress the need for comprehensive tools for identifying CNEs. We present CNEFinder, a tool for identifying CNEs between two given DNA sequences with user-defined criteria. CNEFinder applies the k-mer technique of [56] for computing maximal exact matches. Hence it *does not* require or compute the whole-genome alignment of the two sequences; it *does not* require or compute a whole-genome index such as the suffix array or the BWT (a method of lexicographically sorting all circular rotations of a string, making it easier to compress [20])—see [59], for instance—and it thus finds CNEs from the two sequences directly with user-defined criteria. We have designed CNEFinder in a way that we hope proves useful for the biological community: the tool identifies all CNEs around genes of interest with the aim to facilitate functional experiments. Genome- or chromosome-wide CNE trends may also be revealed as demonstrated by our results. We anticipate that CNEFinder will be a useful tool towards cracking the still largely enigmatic regulatory code of our genome.

5.2 Algorithm CNEFinder

CNEFinder was implemented in the `C++` programming language with OpenMP API for multi-platform shared-memory parallel programming. Our implementation (along with a several-page documentation) is distributed under the GNU General Public License, and is made freely available at `https://github.com/lorrainea/CNEFinder`.

Given two DNA sequences, a reference sequence x and a query sequence y, CNEFinder uses the state-of-the-art k-mer method presented by [56], in conjunction with the well-known *seed and extend* strategy [4, 81, 59], to identify CNEs between x and y. The

DNA sequences are first pre-processed to remove exons and simple and low-complexity repeats from the search, allowing the tool to search for CNEs more accurately. CNEs can then be identified by searching the intergenic and intronic regions around a specific gene as input by the user. CNEs can also be identified through the input of specific index positions of chromosomes that exist in the pair of DNA sequences. Moreover, CNEFinder is able to search for CNEs in entire chromosomes. CNEFinder uses a three-stage approach, specifically tailored for CNE identification:

1. Initially maximal exact matches of a specific length are identified between the specified range in a pair of input sequences;

2. These anchors are then merged to form co-linear sequences of non-overlapping matches;

3. Finally, the matches are extended to the left and right as permits, until a specific threshold or length is reached.

5.2.1 Stage 1: Identifying matches

The k-mer-based method [56] for identifying *maximal exact matches* between two sequences is used to identify exact matches (or anchors) between x and y. These k-mers are exact matches of length k between two sequences, that cannot be extended to the left or the right without causing a mismatch to occur. This algorithm avoids the use of indexing methods, but instead makes use of hash tables. The k-mers of x are first computed using standard bitwise operations; they are then hashed using double hashing; andfinally they are stored. The hash table stores each entry as the value of the k-mer and a list of all positions where the k-mer occurs in x. The corresponding k-mers in y are then matched using the stored hash table. Attempts to extend the matches of all occurrences of stored positions in the table are carried out. These positions are then returned as maximal exact matches.

We measure the identity score between two strings using the simple edit distance model [61]. In this model, the total number of unit-cost edit operations required to transform one string into the other is minimised. The considered operations are insertions, deletions or substitutions of letters. Given a lower bound ℓ on the length of the reported elements and a lower bound $t \in (0, 1]$ on the relative identity threshold between two elements (1 returns identical substrings in x and y), maximal exact matches of minimum length $\lfloor \ell/(\ell - t \times \ell + 1) \rfloor$ are computed via applying the k-mer-based method [56]. This ensures for *exact* matches to be identified, which can then be extended, such that each pair of elements with minimum length ℓ can have an edit distance of at most $\ell - t \times \ell$. This follows from a simple counting argument. The user can alternatively set an explicit value for this minimum length, and then the maximum of the two values is considered for the computation.

Example Take the following factors $x[i\,..\,j]$ of string x, noted as x' and $y[i'\,..\,j']$ of string y, noted as y' and let $\ell = 30$ and $t = 0.7$.

$$x' = \texttt{ATTAC}\boxed{\texttt{GGCG}}\texttt{AATC}\boxed{\texttt{CAAACA}}\texttt{GTGCGGTA}\boxed{\texttt{TTT}}\texttt{GCTGC}$$
$$y' = \texttt{TCGACTA}\boxed{\texttt{GGCG}}\texttt{ACTT}\boxed{\texttt{CAAACA}}\texttt{TGTCGCA}\boxed{\texttt{TTT}}\texttt{TCTCC}$$

k-mers with minimum length $\lfloor 30/(30 - 0.7 \times 30 + 1) = 3$ are identified and stored as anchors.

5.2.2 Stage 2: Merging matches

The anchors found are then merged to produce co-linear sequences of non-overlapping matches and processed further if the combined length of the matches is above a lower bound of nucleotides set by the user with respect to ℓ. The exact identity score at each merging step is calculated by considering the total edit distance of the gaps between the anchors to be merged. The merging process is terminated once the addition of another gap would force the relative identity score to drop below threshold t. For edit-distance

computation, we apply the fast bit-vector algorithm by [75]. Note that this algorithm applies only for simple edit distance.

Example Take the same strings x' and y' as previously presented and values $\ell = 30$ and $t = 0.7$.

$$x' = \texttt{ATTAC}\boxed{\texttt{GGCG}}\texttt{AATC}\boxed{\texttt{CAAACA}}\texttt{GTGCGGTA}\boxed{\texttt{TTT}}\texttt{GCTGC}$$
$$y' = \texttt{TCGACTA}\boxed{\texttt{GGCG}}\texttt{ACTT}\boxed{\texttt{CAAACA}}\texttt{TGTCGCA}\boxed{\texttt{TTT}}\texttt{TCTCC}$$

The gap found at $x'[9\,..\,12]$ and $y'[11\,..\,14]$ has an edit distance (using unit-costs) of 2. The gap found at $x'[19\,..\,26]$ and $y'[21\,..\,27]$ has an edit distance of 4. The current total length of this element is 24, resulting in an identity score of 0.75.

5.2.3 Stage 3: Extending matches

The last stage is to check whether the merged matches can be further extended to the left or to the right. At each step of the extension process, the edit distance of the extension in the left and right direction of the merged matches is computed using Myers' algorithm [75]. The current match is extended in both directions if the threshold allows it or otherwise in the direction having the smallest edit distance. This procedure is repeated until the computed relative identity score of the current length of the match reaches t or when the maximum length u of one of the elements, which is defined by the user, has been reached.

Note that due to the way the extension stage works, the estimated identity score may not be the actual identity score for the whole element: the estimated score could be smaller or equal to the actual. To re-adjust and allow for further extension, the actual identity score is computed for the whole element using Myers' algorithm [75], and the extension process continues accordingly. Those matches that are of length at least ℓ and at most u with relative identity score of at least t are reported as CNEs.

Example Take the same strings x' and y' as previously presented and values $\ell = 30$ and $t = 0.7$.

$$x' = \texttt{ATTAC}\,\boxed{\texttt{GGCGAATCCAAACAGTGCGGTATTT}}\,\texttt{GCTGC}$$
$$y' = \texttt{TCGACTA}\,\boxed{\texttt{GGCGACTTCAAACATGTCGCATTT}}\,\texttt{TCTCC}$$

An extension to the left and right as shown underlined, results in an overall edit distance of 9. The length of the element is 30 which gives a relative identity score of 0.7 which is equal to t. As this element has a length of at least 30, it is reported as a CNE.

5.3 Experimental Results

All datasets and output files referred to in this section can be found at `https://github.com/lorrainea/CNEFinder`. To demonstrate the *accuracy* and *efficiency* of CNEFinder we have conducted the following experiments on a standard desktop PC with an Intel Core i7-4790 CPU at 3.60GHz with 16GB of RAM running a GNU/Linux operating system. All optional parameters were set as default unless stated otherwise.

	200 − 250 bp		250 − 300 bp		300 − 350 bp	
Gene	# CNEs Overlapping	% Nucleotides Overlapping	# CNEs Overlapping	% Nucleotides Overlapping	# CNEs Overlapping	% Nucleotides Overlapping
ZEB2	31/31	84.59	18/18	87.31	20/20	90.36
TSHZ3	35/36	78.49	16/17	80.01	8/8	84.85
EBF3	28/28	87.90	17/17	90.81	16/16	88.21
BCL11A	20/20	81.24	28/28	85.75	14/14	93.61
ZFHX4	18/18	88.02	22/22	89.82	10/10	86.86
	350 − 400 bp		400 − 450 bp		450 − 500 bp	
Gene	# CNEs Overlapping	% Nucleotides Overlapping	# CNEs Overlapping	% Nucleotides Overlapping	# CNEs Overlapping	% Nucleotides Overlapping
ZEB2	14/14	83.17	19/19	91.56	5/5	92.45
TSHZ3	6/6	88.50	12/12	89.36	2/2	90.68
EBF3	6/6	78.46	8/8	83.91	3/3	82.21
BCL11A	10/10	90.73	4/4	83.49	5/5	88.04
ZFHX4	6/6	93.58	5/5	93.10	6/6	87.98

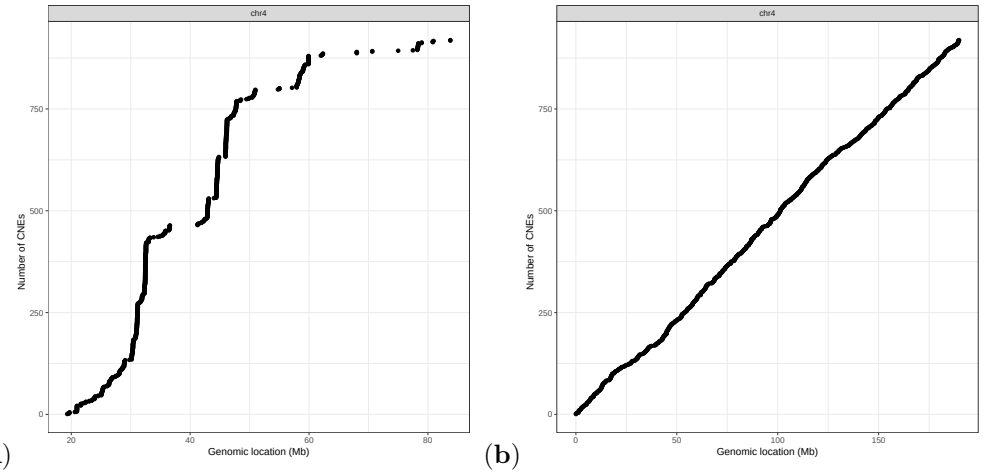
Table 11: CNEs identified for five genes for different length ranges and $t = 95\%$.

5.3.1 CNEFinder against UCNEbase

The first experiment carried out was to identify how accurate CNEFinder is in computing CNEs by comparing against previously identified CNEs stored in the UCNEbase [30], a well-established CNE database. Specifically, this experiment involved identifying CNEs within five different genes between the Human (hg19) and Chicken (galGal3) genomes. Six different length ranges in base pairs (bp) were tested to identify whether CNEFinder obtained the same CNEs as those present in the UCNEbase. A relative identity threshold of $t = 95\%$ was used for all datasets. The results show that CNEFinder identifies almost all elements listed in UCNEbase for these datasets and parameters. Table 11 shows the number of CNEs output by CNEFinder that are overlapping with those stored in the UCNEbase. The table presents these results in the form A/B, where A represents the number of CNEs computed by CNEFinder that were found in the UCNEbase, and B the number of CNEs with a length within the specified range stored in the UCNEbase. We only compute the overlap of the identified CNEs against those in UCNEbase as a precision test as there is no ideal set of CNEs to compare against. This overlap analysis is shown in Table 11 using the average percentage of overlapping nucleotides between the CNEs output by CNEFinder and those stored in the UCNEbase. It was computed using the BEDTools Suite [89]. Note that the majority of CNEs found by CNEFinder were in fact *longer* in length (bp) than those in the UCNEbase, in addition to having a high overlap percentage for all genes at all length ranges. The full list of identified CNEs can be found on-line in the same location as the datasets.

5.3.2 Genomic distribution of CNEs along the chromosome

We also wanted to find out whether the elements returned by CNEFinder are true CNEs in the biological sense. CNEs are known to form clusters in genomes [93], and the distances between consecutive elements follow power-law-like distributions [87, 85]. We plotted the genomic distribution of approximately $1,000$ human CNEs as identified by CNEFinder between Chromosome 4 (chr4) of the Human (hg38) and Chicken (galGal4), with $t = 90\%$



(**a**) (**b**)

Figure 8: Genomic distribution of approximately $1,000$ CNEs along human (hg38) chr4. (**a**) Elements found by CNEFinder. (**b**) CNE-like elements used a control. For more information, see the text.

and $\ell = 50$ bp. As a control, we also plotted the same number of human CNE-like elements; i.e. elements that have one by one, the same length as every CNE in the real set but are distributed randomly on chr4. The function `plotCNEDistribution` from the CNEr R/Bioconductor package [105] was used to produce the plots in Figure 8. Evidently from Figure 8, in the case of elements identified by CNEFinder, many elements are clustered around the same genomic position, while in the case of the control elements, this is clearly the contrary. The latter demonstrates that the elements identified by CNEFinder are indeed CNEs as they display an important biological property.

5.3.3 Efficiency of CNEFinder

We also conducted the following typical runs to demonstrate the time and memory efficiency of CNEFinder. First, we recorded the time taken to compute CNEs with minimum and maximum length (bp), 200 - 250, 250 - 300, 300 - 350, 350 - 400, 400 - 450, and 450 - 500, with $t = 90\%$, between the 143 - 148 Mbp region of Chromosome 2 of the Human (hg19) genome and the 34 - 39 Mbp region of Chromosome 7 of the Chicken (galGal3)

genome using 8 CPU cores. These were, respectively, 4.4s, 4.4s, 4.5s, 4.8s, 4.3s, and 5.2s. The maximum memory used for these runs was 1.6 GB of RAM. Second, we recorded the time taken to compute CNEs with minimum and maximum length 200 - 500 bp with $t = 90\%$ using the whole Chromosome 2 of the human (hg19) genome and the whole Chromosome 7 of the chicken (galGal3) genome using 8 CPU cores. This was 32m30s. The maximum memory used for this run was 5.6 GB of RAM.

5.3.4 Comparison with local-alignment tools

In the last experiment, we exhibit the need for a tool specifically tailored for CNE identification. To this end, we compared CNEFinder to YASS, a state-of-the-art local alignment search tool [77]. YASS works by identifying seeds between a pair of DNA sequences and then extends these matches to *local alignments* between the sequence pair. We ran YASS using regions 76.57 - 79.01 Mbp of Chromosome 8 of the Human (hg19) genome and 123.57 - 124.82 Mbp of Chromosome 2 of the Chicken (galGal3) genome. These are the exact regions used to compute the CNEs for gene ZFHX4 found in Table 11. A dissimilarity threshold of 5% was used to make the experiments as similar as possible. For the elements identified by YASS that did overlap with those in the UCNEbase, the average percentage of overlapping nucleotides was only 31.01%. This can be explained by the optimality criterion of local alignments that does not allow for constraints on the lengths of the alignments. Note that a comparison with other local-alignment tools is beyond the scope of this paper. The rationale of this experiment was to demonstrate the inapplicability of local alignment techniques for CNE identification.

5.4 Conclusion

Due to the lack of published tools for identifying CNEs and the need to systematically investigate their roles in genomes, we have presented CNEFinder, a tool specifically tailored for CNE identification given two DNA sequences. CNEFinder does not require or compute the whole-genome alignment or whole-genome indexes of the two sequences. It thus finds

CNEs from the two sequences directly with user-defined criteria. Experimental results provided here show the accuracy of CNEFinder, compared to existing well-established static databases, as well as its efficiency and ability to reveal biological CNE trends on a chromosome level.

6 Discussion

This thesis presented some of the contributed work towards the field of computational biology, with further applications in image recognition. Specifically, the aim of these algorithms was to identify solutions for sequence comparison whilst avoiding computing alignments under the edit distance measure, where possible. The tool hCED presented in Section 3 provides a heuristic for the computation of the cyclic edit distance. Experimental results presented show the efficiency and accuracy of the tool when provided with both synthetic and real data, compared to existing tools. hCED can be used not only on biological sequences, but also on chain-codes mapped from the orientation of an image. The flexibility of the tool allows it to be used in a digital as well as a biological setting.

As previously stated, the authors in [80] solve the cyclic edit distance problem from an indexing point of view. This is useful for classification, specifically when organisms need be grouped together. It can also be useful for retrieval systems, where the most similar sequence in a given database to an input sequence, in terms of edit distance needs to be identified. hCED can be further explored and manipulated to try and solve this same problem, specifically making use of the β-blockwise q-gram distance measure for a set of strings.

On the other hand, the authors in [7] present new findings for solving the exact circular sequence comparison algorithm under the β-blockwise q-gram distance as studied in [47] (Stage 1 of hCED, algorithm saCSC). They introduce a pre-processing step which employs a number of simple and effective filters allowing for the reduction in the search space. The output of their algorithm is a text of reduced length which can then be input into any circular sequence comparison tool.

They combine their algorithm along with saCSC and compare its performance to saCSC alone. Their implementation has not been made available. However, their experimental results show that when making use of their pre-processing step, saCSC performs in most cases, more than twice as fast as it would than when running alone. Incorporating this

into Stage 1 of hCED can improve the efficiency of the tool while maintaining the accurate results presented here, showing even better performance when compared to the existing cyclic edit distance tools.

Future Directions: Taud et al in [108] present an algorithm Adaboost to identify circular forms in satellite imagery. They make use of classification methods to be able to identify varying circular structures given a number of aerial photographs. These circular geological patterns can be the result of several phenomena including meteoritic, magmatic, tectonic hazards etc. It is clear that circular structures found from these satellite images can fall into one or more categories depending on their shape and structure. These can be classified based on currently known models [94]. Adaboost can be used to identify the circular structures, but the authors present no further details of identifying which class of circular structures each identified object may fall into. The coupling of hCED along with Adaboost can assist with this classification and can help with further analysis of the identified circular structures. Satellite imagery is known to help with predictions and assessments of natural hazards [44], which initially requires the identification of locations of causes, such as volcanic structures. The classification of such structures using hCED can help to assist with such predictions.

As well as predictions in image recognition, hCED can also assist with predictions for applications in molecular biology. For example, it is known that the viral RNA segments of the influenza virus are circular in structure [111]. Influenza can be treated using an inactive strain of the virus and there are also several anti-viral drugs that exist to treat the virus. In order to be able to produce such vaccinations, in depth analysis is required, including looking at the genetic structure of the virus [102]. Research has shown that over time antigenic drifts occur which cause variance and mutations in the seasonal flu [103]. As a result vaccinations need to be continuously adapted to account for these changes. Being able to analyse the molecular structure of the new viruses compared to that of the previous is important where authors in [103], in particular, look at the hamming distance between consecutively released viruses. hCED could greatly aid in this analysis,

specifically looking at the evolution of the circular RNA segments of the virus and also introduce the analysis of the edit distance as well as the current focus on the hamming distance. Future predictions as well as current analysis of antigenic drifts can be carried out allowing for vaccinations to be updated according to these findings.

The tool MARS presented in Section 4 enhances the algorithm of hCED, by providing a tool for the benefits of computing multiple circular sequence alignments. As this tool also makes use of the saCSC algorithm for circular sequence comparison, the pre-processing step presented above [7] can also be used in Stage 1 of MARS. Given a set of d input sequences, the cyclic edit distance computation needs to be applied d^2 times to complete this stage. Making use of the pre-processing filter algorithm can improve the efficiency of MARS while again maintaining accurate results.

The authors in [107] show how MARS has been used in the computation of the complete mitochondrial genome of *Pristurus rupestris rupestris*, a gecko of the Sphaerodactylidae family. The authors show how recent sequencing techniques, including MARS have provided an efficient way to fully sequence the mitochondrial genome related to the Sphaerodactylidae. The circular mitochondrial genomes were rotated using MARS to obtain sequences of the same origin and then aligned to allow for phylogenetic analysis.

Future directions: Not only can MARS assist with the analysis of the evolutionary history of an organism, but it can also help to identiy hereditary traits among humans. It is well known that mitochondria convert energy from food which can be used by cells. This process is called respiration [40]. It is clear from this that mtDNA plays a large role in several processes within the human body. Once of these is metabolism. Tranah et al [112] carried out extensive research on the relation between ethnic groups and their metabolic rates. They claim the proposition that the geographic distribution of mtDNA lineages was the result of adaptation to climate and nutrition. They measured the energy expenditure between African and European haplogroups and found a significant difference between the two groups in terms of resting metabolic rate and energy expenditure. They however carry out no analysis on the mtDNA of the different ethnic groups. MARS can

be used to carry out similar experiments to actually analyse the genetic sequences of the different haplogroups to identify how mtDNA has adapted or mutated according to certain occurrences such as climate change and whether a similar rate of metabolism is the result of closely related mtDNA.

Similarly, [100] presented evidential claims that certain studies showed a correlation between mtDNA mutations and migraine sufferers. Further studies are still required to clarity the significance between migraines and unidentified mutations within mtDNA. As well as this, studies have shown that the fewer the copies of mtDNA, the higher the risk of cardiovascular disease [124]. Further analysis of mtDNA can provide significant evidence to these claims and can help to determine the cause of these genetic diseases. Making use of MARS can help to identify traits in certain groups of people who claim to suffer from similar symptoms. MARS can also help to determine where mutations occur given a set of mtDNA. This analysis can then go on to help make predictions for humans regarding cardiovascular health and migraines as well as allow us to explore the correlation between the characteristics of mtDNA and other diseases.

Section 5 presented a tool for finding conserved non-coding elements in genomes. The lack of publicly available tools for the computation of CNEs restricts the ability to analyse their behaviour and functionality. CNEFinder is the first publicly available tool, tailored specifically for the computation of CNEs. This breakthrough allows for the speed and ease of identifying CNEs in genomes using user-defined parameters, without computing whole genome alignments or indexes.

Although experimental results show high accuracy of the tool when comparing to static databases, there are still further improvements to enhance the flexibility and performance of the tool. When an extension to the left and right of the merged matches has the same edit distance score, the current extension method chooses a random direction to extend in, if extending in both the left and right directions individually are below the threshold but extending in both directions together would exceed the threshold. This results in CNEs possibly being of shorter length and therefore below ℓ, the minimum CNE length

and so not reported. Extension techniques discussed in [1] show that given a group of merged seeds of length $m < \ell$, an extension of length $2 \times \ell - m$ to both the left and right direction allows all CNE elements of at least length ℓ and a relative identity score t to be identified [99].

In addition to this, the purpose of the tool is to identify CNE elements conserved across evolution and as previously stated several organisms share common CNEs. CNEFinder currently takes as input a pair of genomes and computes CNEs identified between the two sequences. [14] states that 481 elements longer than 200bp are absolutely conserved within the human, rat, and mouse genomes. On top of this, the majority of these elements are also conserved in the chicken and dog genomes as well as in fish. Clearly a need to identify CNEs in several genomes simultaneously would enhance CNEFinder and allow for a more in depth analysis of the functionality of these elements.

Future directions: Being able to analyse these elements efficiently can allow for a better understanding of what causes CNEs and what role they play during cell division. One important query raised in recent research is the role of CNEs in disease pathogenesis [86]. Understanding the correlation between CNEs and topologically associating domains (TADs), genomic regions which increase sequence interaction, can help us understand more about distinct developmental disorders and cancer. Research has show that the disruption of boundaries of TADs is associated with limb phenotypes [65]. The focus of this research analysed mutations of TAD boundaries to identify how this can contribute to disease etiology. Identifying the relationship between CNEs and TADs can help to answer these queries in understanding how or if they play a role in TAD formation. This analysis could take longer than necessary due to the lack of existing tools for CNE idenification. Making use of CNEFinder to identify these CNEs could allow more extensive research to be carried out on the functionality of CNEs rather than spending a lot of time identifying suitable CNE given a genetic sequence.

Overall, through experimental analysis, these tools have proved to provide a strong contribution towards the field of computational biology and image recognition. Nonethe-

less, there are a range of algorithmic problems that remain unsolved within these fields. This thesis provides an introduction to some of the computational problems available and suitable methods to solve them and provides an insight into how these tools can be further used to enhance their existence.

References

[1] N. Ahmed, K. Bertels, and Z. Al-Ars. A comparison of seed-and-extend techniques in modern DNA read alignment algorithms. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1421–1428, Dec 2016.

[2] N. Ahsan and K. V. Shah. *Polyomaviruses and Human Diseases*, pages 1–18. Springer New York, New York, NY, 2006.

[3] T. Allers and M. Mevarech. Archaeal genetics — the third way. *Nature Reviews Genetics*, 6:58–73, 2005.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[5] S. Aparicio, A. Morrison, A. Gould, J. Gilthorpe, C. Chaudhuri, P. Rigby, R. Krumlauf, and S. Brenner. Detecting conserved regulatory elements with the model genome of the japanese puffer fish, fugu rubripes. *Proceedings of the National Academy of Sciences*, 92(5):1684–1688, 1995.

[6] T. Athar, C. Barton, W. Bland, J. Gao, C. S. Iliopoulos, C. Liu, and S. P. Pissis. Fast circular dictionary-matching algorithm. *Mathematical Structures in Computer Science*, FirstView:1–14, 2015.

[7] Md.A.R. Azim, M. Kabir, and M.S. Rahman. A simple, fast, filter-based algorithm for circular sequence comparison. In M.S. Rahman, W. Sung, and R. Uehara, editors, *WALCOM: Algorithms and Computation*, pages 183–194, Cham, 2018. Springer International Publishing.

[8] P. D. Baas. DNA replication of single-stranded escherichia coli DNA phages. *Biochimica et Biophysica Acta (BBA) - Gene Structure and Expression*, 825(2):111–139, 1985.

[9] I. A. Babarinde and N. Saitou. Genomic locations of conserved noncoding sequences and their proximal protein-coding genes in mammalian expression dynamics. *Molecular Biology and Evolution*, 33(7):1807–1817, 2016.

[10] S. Barrachina and A. Marzal. Speeding up the computation of the edit distance for cyclic strings. *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, 2:891–894, 2000.

[11] C. Barton, C. S. Iliopoulos, R Kundu, S. P. Pissis, A. Retha, and F. Vayani. Accurate and efficient methods to improve multiple circular sequence alignment. In E. Bampis, editor, *Experimental Algorithms - 14th International Symposium, SEA 2015, Proceedings*, volume 9125 of *Lecture Notes in Computer Science*, pages 247–258. Springer, 2015.

[12] C. Barton, C. S. Iliopoulos, and S. P. Pissis. Fast algorithms for approximate circular string matching. *Algorithms for Molecular Biology*, 9:9, 2014.

[13] C. Barton, C. S. Iliopoulos, and S. P. Pissis. Average-case optimal approximate circular string matching. In A.-H. Dediu, E. Formenti, C. Martin-Vide, and B Truthe, editors, *Language and Automata Theory and Applications*, volume 8977 of *Lecture Notes in Computer Science*, pages 85–96. Springer Berlin Heidelberg, 2015.

[14] G. Bejerano, M. Pheasant, I. Makunin, S. Stephen, W. J. Kent, J. S. Mattick, and D. Haussler. Ultraconserved elements in the human genome. *Science*, 304(5675):1321–1325, 2004.

[15] E. M. Blackwood and J. T. Kadonaga. Going the distance: A current view of enhancer action. *Science*, 281(5373):60–63, 1998.

[16] M. Blanchette, W. J. Kent, C. Riemer, L. Elnitski, A. F. A. Smit, K. M. Roskin, R. Baertsch, K. Rosenbloom, H. Clawson, E. D. Green, D. Haussler, and W. Miller. Aligning Multiple Genomic Sequences With the Threaded Blockset Aligner. *Genome Research*, 14(4):708–715, 2004.

[17] P. Bonizzoni and G. D. Vedova. The complexity of multiple sequence alignment with sp-score that is a metric. *Theoretical Computer Science*, 259(1):63–79, 2001.

[18] R. Brodie, A. J. Smith, R. L. Roper, V. Tcherepanov, and C. Upton. Base-By-Base: Single nucleotide-level analysis of whole viral genome alignments. *BMC Bioinform*, 5(1):96, 2004.

[19] H. Bunke and U. Buhler. Applications of approximate string matching to 2D shape recognition. *Pattern Recognition*, 26:1797–1812, 1993.

[20] M. Burrows and D.J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, 1994.

[21] E. Cambouropoulos, T. Crawford, and C.S. Iliopoulos. Pattern processing in melodic sequences: Challenges, caveats and prospects. *Computers and the Humanities*, 35(1):9–21, 2001.

[22] S. Capella-Gutierrez, J. M. Silla-Martinez, and T. Gabaldon. trimAl: a tool for automated alignment trimming in large-scale phylogenetic analyses. *Bioinformatics*, 25:1972–1973, 2009.

[23] J. Chang, P. D. Tommaso, and C. Notredame. TCS: A new multiple sequence alignment reliability measure to estimate alignment accuracy and improve phylogenetic tree reconstruction. *Molecular Biology and Evolution*, 2014.

[24] M. Chatzou, C. Magis, J. Chang, C. Kemena, G. Bussotti, I. Erb, and C. Notredame. Multiple sequence alignment modeling: methods and applications. *Briefings in Bioinformatics*, pages 1–15, 2015.

[25] D. J. Craik and N. M. Allewell. Thematic minireview series on circular proteins. *The Journal Of Biological Chemistry*, 287:26999–27000, 2012.

[26] A. Criscuolo and S. Gribaldo. BMGE (block mapping and gathering with entropy): a new software for selection of phylogenetic informative regions from multiple sequence alignments. *BMC Evolutionary Biology*, 10:1–21, 2010.

[27] M. Crochemore, G. Fici, R. Mercas, and S. P. Pissis. Linear-time sequence comparison using minimal absent words & applications. In E. Kranakis, G. Navarro, and E. Chávez, editors, *LATIN 2016: Theoretical Informatics: 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, Lecture Notes in Computer Science, pages 334–346. 2016.

[28] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*. Cambridge University Press, New York, NY, USA, 2007.

[29] F.J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7:171–176, 1964.

[30] S. Dimitrieva and P. Bucher. UCNEbase–a database of ultraconserved non-coding elements and genomic regulatory blocks. *Nucleic Acids Research*, 41(D1):D101–D109, 2013.

[31] A. Dousse, T. Junier, and E. M. Zdobnov. CEGA–a catalog of conserved elements from genomic alignments. *Nucleic Acids Research*, 44(D1):D96–D100, 2015.

[32] A. W. M. Dress, C. Flamm, G. Fritzsch, S. Grünewald, M. Kruspe, S. J. Prohaska, and P. F. Stadler. Noisy: Identification of problematic columns in multiple sequence alignments. *Algorithms for Molecular Biology*, 3:1–10, 2008.

[33] I. Dubchak, M. Brudno, G. G. Loots, L. Pachter, C. Mayor, E. M. Rubin, and K. A. Frazer. Active conservation of noncoding sequences revealed by three-way species comparisons. *Genome Research*, 10(9):1304–1306, 2000.

[34] R. C Edgar. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, 5:1–19, 2004.

[35] R. C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32:1792–1797, 2004.

[36] P. G. Engström, D. Fredman, and B. Lenhard. Ancora: a web resource for exploring highly conserved noncoding elements and their association with developmental regulatory genes. *Genome Biology*, 9(2):R34, 2008.

[37] F. Fernandes, L. Pereira, and A. T. Freitas. CSA: an efficient algorithm to improve circular DNA multiple alignment. *BMC Bioinformatics*, 10:1–13, 2009.

[38] W. M. Fitch. Distinguishing homologous from analogous proteins. *Systematic Biology*, 19(2):99–113, 1970.

[39] W. Fletcher and Z. Yang. INDELible: a flexible simulator of biological sequence evolution. *Molecular Biology and Evolution*, 8:1879–1888, 2009.

[40] F. Fontanesi. *Mitochondria: Structure and Role in Respiration*, pages 1–13. American Cancer Society, 2015.

[41] H. Freeman. On the encoding of arbitrary geometric configurations. *Electronic Computers, IRE Transactions on*, EC-10:260–268, 1961.

[42] G. Fritzsch, M. Schlegel, and P.F. Stadler. Alignments of mitochondrial genome arrangements: Applications to metazoan phylogeny. *Journal of Theoretical Biology*, 240(4):511–520, 2006.

[43] G. Fritzscha, M. Schlegela, and P.F. Stadlera. Alignments of mitochondrial genome arrangements: Applications to metazoan phylogeny. *Journal of Theoretical Biology*, 240:511–520, 2005.

[44] T. W. Gillespie, J. Chu, E. Frankenberg, and D. Thomas. Assessment and prediction of natural hazards from satellite imagery. *Progress in Physical Geography: Earth and Environment*, 31(5):459–470, 2007. PMID: 25170186.

[45] R. C. Gonzalez and R. E. Woods. *Digital Image Processing (2nd Edition)*. Prentice Hall, 2002.

[46] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, 1982.

[47] R. Grossi, C. S. Iliopoulos, R. Mercas, N. Pisanti, S. P. Pissis, A. Retha, and F. Vayani. Circular sequence comparison: algorithms and applications. *Algorithms for Molecular Biology*, 11:12, 2016.

[48] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[49] R.S. Harris. *Improved Pairwise Alignment of Genomic DNA*. PhD thesis, Pennsylvania State University, University Park, PA, USA, 2007.

[50] D. R. Helinski and D. B. Clewell. Circular DNA. *Annual Review of Biochemistry*, 40:899–942, 1971.

[51] P. Hogeweg and B. Hesper. The alignment of sets of sequences and the construction of phyletic trees: An integrated method. *Journal of Molecular Evolution*, 20(2):175–186, 1984.

[52] A. K. Jain and E. Angel. Image restoration, modelling, and reduction of dimensionality. *IEEE Transactions on Computers*, C-23(5):470–476, May 1974.

[53] P. Jokinen and E. Ukkonen. Two algorithms for approximate string matching in static texts. In *MFCS*, pages 240–248, 1991.

[54] T.H. Jukes and C.R. Cantor. *Evolution of Protein Molecules.* Academy Press, New York, 1969.

[55] H. Kasamatsu and J. Vinograd. Replication of circular DNA in eukaryotic cells. *Annual Review of Biochemistry*, 43:695–719, 1974.

[56] N. Khiste and L. Ilie. E-MEM: efficient computation of maximal exact matches for very large genomes. *Bioinformatics*, 31(4):509–514, 2015.

[57] P. Kuck, K. Meusemann, J. Dambach, B. Thormann, B. M. von Reumont, J. W. Wagele, and B. Misof. Parametric and non-parametric masking of randomness in sequence alignments can be improved and leads to better resolved trees. *Frontiers in Zoology*, 7:1–12, 2010.

[58] S. Kumar and A. Filipski. Multiple sequence alignment: in pursuit of homologous DNA positions. *Genome Research*, 17(2):127–135, 2007.

[59] S. Kurtz, A. Phillippy, A. L. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S. L. Salzberg. Versatile and open software for comparing large genomes. *Genome Biology*, 5(2):R12, Jan 2004.

[60] Y. LeCun and C. Cortes. The MNIST database of handwritten digits. 1999.

[61] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[62] S. Lockton and B. S. Gaut. Plant conserved non-coding sequences and paralogue evolution. 21(1):60–65, 2005. Exported from https://app.dimensions.ai on 2018/12/08.

[63] H. Lodish. *Molecular Cell Biology (4th edition)*. New York: W.H. Freeman and Co, 2013.

[64] V. Lomonaco, R. Martoglia, F. Mandreoli, L. Anderlucci, W. Emmett, S. Bicciato, and C. Taccioli. UCbase 2.0: ultraconserved sequences database (2014 update). *Database*, 2014, 2014.

[65] D. G. Lupiáñez, K. Kraft, V. Heinrich, P. Krawitz, F. Brancati, E. Klopocki, D. Horn, H. Kayserili, J. M. Opitz, R. Laxova, F. Santos-Simarro, B. Gilbert-Dussardier, L. Wittler, M. Borschiwer, S. A. Haas, M. Osterwalder, M. Franke,

B. Timmermann, J. Hecht, M. Spielmann, A. Visel, and S. Mundlos. Disruptions of topological chromatin domains cause pathogenic rewiring of gene-enhancer interactions. *Cell*, 161(5):1012 – 1025, 2015.

[66] M. Maes. On a cyclic string-to-string correction problem. *Information Processing Letters*, 35(2):73–78, 1990.

[67] M. Maes. Polygonal shape recognition using string-matching techniques. *Pattern Recognition*, 24(5):433–440, 1991.

[68] U. Manber and E. W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[69] A. Marzal and V. Palazon-Gonzalez. Dynamic time warping of cyclic strings for shape matching. In *Pattern Recognition and Image Analysis*, volume 3687 of *Lecture Notes in Computer Science*, pages 644–652. Springer, 2005.

[70] A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15:926–932, 1993.

[71] G. P. McCormack and J. P. Clewley. The application of molecular phylogenetics to the analysis of viral genome diversity and evolution. *Reviews in Medical Virology*, 12(4):221–238.

[72] R. A. Mollineda, E. Vidal, and F. Casacuberta. Efficient techniques for a very accurate measurement of dissimilarities between cyclic patterns. *Advances in Pattern Recognition*, 1876:337–346, 2000.

[73] R. A. Mollineda, E. Vidal, and F. Casacuberta. Cyclic sequence alignments: Approximate versus optimal techniques. *International Journal of Pattern Recognition and Artificial Intelligence*, 16:291–299, 2002.

[74] A. Mosig, I. L. Hofacker, and P. F. Stadler. Comparative analysis of cyclic sequences: Viroids and other small circular RNAs. In R Giegerich and J Stoye, editors, *Lecture Notes in Informatics*, pages 93–102. Proceedings GCB, 2006.

[75] G. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM (JACM)*, 46:395–415, 1999.

[76] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[77] L. Noé and G. Kucherov. YASS: enhancing the sensitivity of DNA similarity search. *Nucleic Acids Research*, 33(suppl_2):W540–W543, 2005.

[78] C. Notredame, D. G. Higgins, and J. Heringa. T-coffee: a novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1):205–217, 2000.

[79] V. Palazon-Gonzalez and A. Marzal. On the dynamic time warping of cyclic sequences for shape retrieval. *Image and Vision Computing*, 30:978–990, 2012.

[80] V. Palazon-Gonzalez and A. Marzal. Speeding up the cyclic edit distance using laesa with early abandon. *Pattern Recognition Letters*, 62:1–7, 2015.

[81] W. R. Pearson. Flexible sequence similarity searching with the FASTA3 program package. *Methods Mol Biol.*, 132:185–219, 2000.

[82] O. Penn, E. Privman, H. Ashkenazy, G. Landan, D. Graur, and T. Pupko. GUIDANCE: a web server for assessing alignment confidence scores. *Nucleic Acids Research*, 38(suppl 2):23–28, 2010.

[83] J. Persampieri, D. I. Ritter, D. Lees, Q. Lehoczky, J.and Li, S. Guo, and J. H. Chuang. cneViewer: a database of conserved non-coding elements for studies of tissue-specific gene regulation. *Bioinformatics*, 24(20):2418–2419, 2008.

[84] A. Phillips, D. Janies, and W. Wheeler. Multiple sequence alignment in phylogenetic analysis. *Molecular Phylogenetics and Evolution*, 16(3):317–330, 2000.

[85] D. Polychronopoulos, L. Athanasopoulou, and Y. Almirantis. Fractality and entropic scaling in the chromosomal distribution of conserved noncoding elements in the human genome. *Gene*, 584(2):148–160, 2016.

[86] D. Polychronopoulos, J. W. D. King, A. J. Nash, G. Tan, and B. Lenhard. Conserved non-coding elements: developmental gene regulation meets genome organization. *Nucleic Acids Research*, page gkx1074, 2017.

[87] D. Polychronopoulos, D. Sellis, and Y. Almirantis. Conserved noncoding elements follow power-law-like distributions in several genomes as a result of genome dynamics. *PLOS ONE*, 9(5):1–12, 05 2014.

[88] D. Polychronopoulos, E. Weitschek, S. Dimitrieva, P. Bucher, G. Felici, and Y. Almirantis. Classification of selectively constrained DNA elements using feature vectors and rule-based classifiers. *Genomics*, 104(2):79–86, 2014.

[89] A. R. Quinlan and I. M. Hall. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842, 2010.

[90] C. R. Rao. Geometry of circular vectors and pattern recognition of shape of a boundary. *Proceedings of the National Academy of Sciences*, 95(22):12783–12786, 1998.

[91] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4:406–425, 1987.

[92] A. Sandelin, W. Alkema, P. Engström, W. W. Wasserman, and B. Lenhard. JASPAR: an open-access database for eukaryotic transcription factor binding profiles. *Nucleic Acids Research*, 32(suppl_1):D91–D94, 2004.

[93] A. Sandelin, P. Bailey, S. Bruce, P. G. Engström, J. M. Klos, W. W. Wasserman, J. Ericson, and B. Lenhard. Arrays of ultraconserved non-coding regions span the loci of key developmental genes in vertebrate genomes. *BMC Genomics*, 5(1):99, 2004.

[94] J.-Y Scanvic and J-P. Deroin. *Aerospatial remote sensing in geology.* 01 1997.

[95] S. Schwartz, W. J. Kent, A. Smit, Z. Zhang, R. Baertsch, R. C. Hardison, D. Haussler, and W. Miller. Human-mouse alignments with BLASTZ. *Genome Research*, 13(1):103–107, 2003.

[96] F. Sievers, A. Wilm, D. Dineen, T. J. Gibson, K. Karplus, W. Li, R. Lopez, H. McWilliam, M. Remmert, J. Söding, J. D. Thompson, and D.G. Higgins. Fast, scalable generation of high-quality protein multiple sequence alignments using clustal omega. *Molecular systems biology*, 7:539, 2011.

[97] T. Sikora. The MPEG-7 visual standard for content description-an overview. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11:696–702, 2001.

[98] V. A. Simossis and J. Heringa. Integrating protein secondary structure prediction and multiple sequence alignment. *Current Protein & Peptide Science*, 5(4):249–266, 2004.

[99] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[100] M. Sparaco, M. Feleppa, R. B. Lipton, A. M. Rapoport, and M. E. Bigal. Mitochondrial dysfunction and migraine: Evidence and hypotheses. *Cephalalgia*, 26(4):361–372, 2006. PMID: 16556237.

[101] A. Stamatakis. RAxML Version 8: A tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30:1312–1313, 2014.

[102] G. Stiver. The treatment of influenza with antiviral drugs. *CMAJ*, 168(1):49–57, 2003.

[103] O. Suptawiwat, A. Kongchanagul, C. Boonarkart, and P. Auewarakul. H1N1 seasonal influenza virus evolutionary rate changed over time. *Virus Research*, 250:43 – 50, 2018.

[104] G. Talavera and J. Castresana. Improvement of phylogenies after removing divergent and ambiguously aligned blocks from protein sequence alignments. *Systematic Biology*, 56(4):564–577, 2007.

[105] G. Tan. CNEr. http://bioconductor.org/packages/release/bioc/html/CNEr.html, 2017.

[106] G. Tan, M. Muffato, C. Ledergerber, J. Herrero, N. Goldman, M. Gil, and C. Dessimoz. Current methods for automated filtering of multiple sequence alignments frequently worsen single-gene phylogenetic inference. *Systematic Biology*, 64(5):778–791, 2015.

[107] P. Tarroso, M. Simó-Riudalbas, and S. Carranza. The complete mitochondrial genome of pristurus rupestris rupestris. *Mitochondrial DNA Part B*, 2(2):802–803, 2017.

[108] H. Taud, J. C. Herrera-Lozada, J. A. álvarez-Cedillo, M. Marciano-Melchor, R. Silva-Ortigoza, and M. Olguín-Carbajal. Circular object recognition from satellite images. In *2012 IEEE International Geoscience and Remote Sensing Symposium*, pages 2324–2327, July 2012.

[109] M. Thanbichler, S. C. Wang, and L. Shapiro. The bacterial nucleoid: A highly organized and dynamic structure. *Journal of Cellular Biochemistry*, 96(3):506–521.

[110] J. D. Thomson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting,

position-specific gap penalties and weight matrix choice. *Nucleic Acids Ressearch*, 22:4673–4680, 1994.

[111] D. M. Tralli, R. G. Blom, V. Zlotnicki, A. Donnellan, and D. L. Evans. Satellite remote sensing of earthquake, volcano, flood, landslide and coastal inundation hazards. *ISPRS Journal of Photogrammetry and Remote Sensing*, 59(4):185 – 198, 2005. Remote Sensing and Geospatial Information for Natural Hazards Characterization.

[112] G. J. Tranah, T. M. Manini, K. K. Lohman, M. A. Nalls, S. Kritchevsky, A. B. Newman, T. B. Harris, I. Miljkovic, A. Biffi, S. R. Cummings, and Y. Liu. Mitochondrial dna variation in human metabolic rate and energy expenditure. *Mitochondrion*, 11(6):855 – 861, 2011.

[113] E. Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92:191–211, 1992.

[114] S. Vinga and J. Almeida. Alignment-free sequence comparison—a review. *Bioinformatics*, 19(4):513–523, 2003.

[115] A. Visel, S. Minovitsky, I. Dubchak, and L. A. Pennacchio. VISTA Enhancer Browser–a database of tissue-specific human enhancers. *Nucleic Acids Research*, 35(suppl_1):D88–D92, 2006.

[116] G. Wang and R. L. Dunbrack. Scoring profile-to-profile sequence alignments. *Protein Science*, 13(6):1612–1626, 2004.

[117] L. Wang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1:337–348, 1994.

[118] M. Warnefors, B. Hartmann, S. Thomsen, and C. R. Alonso. Combinatorial gene regulatory functions underlie ultraconserved elements in Drosophila. *Molecular Biology and Evolution*, 33(9):2294–2306, 2016.

[119] R. Weil and J. Vinograd. The cyclic helix and cyclic coil forms of polyoma viral dna. *Proceedings of the National Academy of Sciences*, 50(4):730–738, 1963.

[120] J. Weiner and E. Bornberg-Bauer. Evolution of circular permutations in multidomain proteins. *Mol Biol Evol*, 23(4):734–743, 2006.

[121] A. Woolfe, D. K. Goode, J. Cooke, H. Callaway, S. Smith, P. Snell, G. K. McEwen, and G. Elgar. CONDOR: a database resource of developmentally associated conserved non-coding elements. *BMC Developmental Biology*, 7(1):100, 2007.

[122] M. Wu, S. Chatterji, and J. A. Eisen. Accounting for alignment uncertainty in phylogenomics. *PLoS ONE*, 7:1–10, 01 2012.

[123] J. Xiong. *Essential Bioinformatics*. Cambridge University Press, Texas A&M University, 2006. Cambridge Books Online.

[124] P. Yue, S. Jing, L. Liu, F. Ma, Y. Zhang, C. Wang, H. Duan, K. Zhou, Y. Hua, G. Wu, and Y. Li. Association between mitochondrial dna copy number and cardiovascular disease: Current evidence based on a systematic review and meta-analysis. *PLOS ONE*, 13(11):1–15, 11 2018.

[125] M. Šošić and M. Šikić. Edlib: a C/C++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, 33(9):1394–1395, 2017.